2014

SVM-BASED VOLITIONAL ARTIFICIAL LEG CONTROL VIA UBIQUITOUS SMALL AND LOW POWER ARCHITECTURES

Robert Hernandez
University of Rhode Island, brooklynboy@cox.net

SVM-BASED VOLITIONAL ARTIFICIAL LEG CONTROL VIA UBIQUITOUS SMALL AND LOW POWER ARCHITECTURES

BY

ROBERT HERNANDEZ

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

COMPUTER ENGINEERING

UNIVERSITY OF RHODE ISLAND

2014

DOCTOR OF PHILOSOPHY DISSERTATION

OF

ROBERT HERNANDEZ

APPROVED:

Dissertation Committee:

Major Professor Qing Yang

Jien-Chung Lo

Manbir Sodhi

Nassar H. Zawia
DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

ABSTRACT

Presented within this dissertation is the evolution of the research leading to the selection of small, low power architectural solutions to the University of Rhode Island's (URI) Neural Machine Interface (NMI) algorithm. The NMI is designed to provide volitional control of an artificial limb for transfemoral amputees. The NMI algorithm is based on neuromuscular–mechanical fusion, gait phase dependent, non-linear support vector machine (SVM) classification. URI's NMI algorithm utilizes electromyography to detect direct commands from the human brain to the residual thigh muscles in conjunction with mechanical signals derived from loadcell to determine the user's intended locomotion mode.

Of utmost importance is the classification accuracy, since any misclassification can cause the user to stumble, possibly leading to serious injury or death. Furthermore, of importance is the development of a small and low power architectural solution, such that it can be included within the confines of the artificial limb. URI has tackled both these challenges, leading to its mobile Central Processing Unit (CPU) solution. The mobile CPU solution was the first solution with sufficient processing throughput to execute the NMI at 20ms window increments. This led to a steady state classification accuracy of 99.94%, during real-time testing, with an able bodied subject. This testing included a total of 14000+ static classifications, and is currently URI's only, 20ms window increment, state of the art algorithmic and architectural solution to undergo real time human subject testing and evaluation.

In contrast to URI's NMI algorithm, other state of the art algorithms provide volitional control through either echo control or solely thru intrinsic mechanical feedback. In echo control, sensors are placed within the sound leg to determine the intended locomotion mode. In most cases these sensors typically communicate wirelessly with the artificial limb to provide the feedback necessary for volitional control. This approach is disadvantaged in the fact that it requires that sensors be instrumented on the sound limb, the user must always lead with the sound limb, and the wireless communications may possibly be jammed. Current algorithms based solely on intrinsic mechanical feedback, have been shown to provide high accuracy, but have had difficulty dealing with more than two simultaneous dynamic locomotion modes (e.g. - walk, stair up, stair down, ramp up, and ramp down).

Clearly URI's NMI solution has advantages over other state of the art powered lower limb prosthetic control algorithms. It provides volitional control without the need to instrument the sound limb, without the need of wireless communications, can easily detect at least seven simultaneous locomotion modes, provides smooth and highly responsive locomotion transition detection and does so with high accuracy. This accuracy can be attributed to the use of neuromuscular-mechanical fusion, SVM detection and 20ms window analysis increments. URI's small, low power, architectural solutions are leading the way towards highly accurate volitional artificial leg control of powered prosthetic devices, thereby making a bionic leg a feasible reality in the near future.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Qing Yang for his guidance, support, and most importantly his understanding and patience throughout the duration of the research necessary to make this dissertation a reality. His spectacular ability and skill in writing academic papers is something I will strive to achieve in the future. With every edit that Dr. Yang performed, my papers' quality increased. I will forever be grateful for his willingness to take me under his wing and show me how to perform quality Ph.D. research and write quality Ph.D. papers.

In addition to Dr. Yang, I would like to thank my dissertation committee: Professors J.C. Lo, Manbir Sodhi, Yan (Lindsay) Sun, and Philip Datseris for reviewing the manuscripts within this dissertation, for taking the time to serve and participate in my comprehensive examination and dissertation defense.

I also want to thank Dr. He Huang whom, along with Dr. Yang, helped me to write Ph.D. level papers. I had never met Dr. Huang prior to proposing the SVM C-based systems. Although, initially skeptical on the feasibility of achieving real time performance, she gave me the opportunity to prove myself and for that I will be forever grateful to her. In addition I would like to thank Dr. Haibo He, whom consistently participated in the weekly group reviews and always provided excellent input. As an SVM expert, which I am not, it was always encouraging when Dr. He confirmed or agreed with my speculation and/or findings; it definitely let me know I was heading in the correct direction.

I am also grateful to other faculty and staff members of the Electrical, Computer, and Biomedical Engineering department. In particular, I am grateful to Dr.

Resit Sendag, Dr. Godi Fischer, Dr. Richard Vaccaro, Dr. Frederick Vetter, and Ms. Meredith Leach Sanders. I would also like to extend my thanks to Fan Zhang, Xiaorong Zhang and my other friends and colleagues. A special thank you goes out to the NUWC Code 852 “Think Tank” (Jason Kane, Matt Seaton, and Will Simoneau) for proof reading my papers, making recommendations, but more importantly for ALWAYS being there when I needed them most.

I would like to express my gratitude to Dr. Pierre Corriveau, Dr. Anthony Ruffa and Neil Dubois, which helped fund the vast majority of my research. Dr. Corriveau said I would never finish my research, if I were doing it strictly on my own personal time. He was definitely correct and I appreciate his guidance and help in finding NUWC and ONR opportunities to fund my research part time.

Special thanks go out to my NUWC management, whom happen to be really good friends, Dr. Brian McKeon, Hector Lopez, John Fastino and Dan Freitas. Without their constant support and guidance, I would have given up within the first year. Their willingness to give me up half time to allow me to fulfill my educational goals is appreciated greatly. Furthermore, I am grateful to them for helping me get back on my horse, over and over and over again. A large portion of this accomplishment is clearly and obviously due to their support, guidance, and friendship.

Last, but not least by any means, I would like to thank my wife, Debra, my daughter, Patria, and my son, Ricky. Behind every man is a much better woman. I sold this educational endeavor of mine to my wife, back in 2008, as a maximum 2-year part time effort to get my Master’s degree. Here I am in 2014, writing a dissertation in

partial fulfillment of my Ph.D. in Electrical Engineering. During the past six years, my daughter has graduated from high school, entered college and is currently in her junior year. During the past six years, my son has graduated from middle school and has just entered high school. Of course, I've spent the majority of our family time doing school work and performing research. My wife has been a very busy woman with the family. There are no words to express the gratitude and appreciation I have for my wife. I want to apologize for my absence during the last six years, I thank you for your understanding of my educational goals and for your support in helping me meet my educational goals. Deb, you are my love of a lifetime.

PREFACE

This dissertation is written in the manuscript format. It consists of four manuscripts organized as follows:

Manuscript 1:

Robert Hernandez, Fan Zhang, Xiaorong Zhang, He Huang, and Qing Yang, "Promise of a Low Power Mobile CPU Based Embedded System in Artificial Leg Control," published in *the proceedings of the 34th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC '12)*, San Diego, CA, 2012. pp. 5250-5253.

Manuscript 2:

Robert Hernandez and John Faella, "Towards Policy and Guidelines for the Selection of Computational Engines," published in *the proceedings of the 7th Annual IEEE Systems Conference (SysCon '13)*, Orlando, FL, 2013. pp. 88-95.

Manuscript 3:

Robert Hernandez, Jason Kane, Fan Zhang, Xiaorong Zhang, and He Huang, "Towards Ubiquitous Mobile-Computing-Based Artificial Leg Control," submitted to *IEEE Transactions on Mobile Computing*.

Manuscript 4:

Robert Hernandez, Qing Yang, He Huang, Fan Zhang and Xiaorong Zhang, "Design and Implementation of a Low Power Mobile CPU Based Embedded System for Artificial Leg Control," published *in the proceedings of the 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC '13)*, Osaka, Japan, 2013. pp. 5769-5772.

This dissertation is concluded and suggestions on further development of the Neural Machine Interface algorithm and hardware architecture are provided in Chapter 5.

TABLE OF CONTENTS

LIST OF TABLES

TABLE PAGE

LIST OF FIGURES

FIGURE PAGE

MANUSCRIPT 1

Promise of a Low Power Mobile CPU based Embedded System in Artificial Leg Control

by

[1] Robert Hernandez, Fan Zhang, Xiaorong Zhang, He Huang and Qing Yang

is published in *the proceedings of the 34th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC '12)*,

San Diego, CA, 2012. p. 5250-5253.

[1] Robert Hernandez, Fan Zhang, Xiaorong Zhang, He Huang and Qing Yang are with Department of Electrical, Computer, and Biomedical Engineering, University of Rhode Island, Kingston, RI, 02881, Email {rhernandez, fzhang, zxiaorong, huang, qyang}@ele.uri.edu.

Abstract

This paper presents the design and implementation of a low power embedded system using mobile processor technology (Intel Atom™ Z530 Processor) specifically tailored for a neural-machine interface (NMI) for artificial limbs. This embedded system effectively performs our previously developed NMI algorithm based on neuromuscular-mechanical fusion and phase-dependent pattern classification. The analysis shows that NMI embedded system can meet real-time constraints with high accuracies for recognizing the user's locomotion mode. Our implementation utilizes the mobile processor efficiently to allow a power consumption of 2.2 watts and low CPU utilization (less than 4.3%) while executing the complex NMI algorithm. Our experiments have shown that the highly optimized C program implementation on the embedded system has superb advantages over existing PC implementations on MATLAB. The study results suggest that mobile-CPU-based embedded system is promising for implementing advanced control for powered lower limb prostheses.

1.1 Introduction

A neural-machine interface (NMI) based on neuromuscular-mechanical fusion [1] and phase-dependent pattern recognition (PR) strategy [2] has been successfully developed in our research group to identify user intent for volitional control of powered lower limb prostheses. Embedded implementation of this complex NMI algorithm for real-time operation is essential for lower limb prostheses, but is challenging due to the rigorous system requirements. First, the prosthesis control must be accurate and responsive to enable lower limb amputees to perform different tasks safely and intuitively. In addition, the prosthesis control system must perform continuously for 6-8 hours daily without interruption. Finally, the system must be easily integrated into the prosthetic limb. These requirements demand the embedded system to be computational powerful, low power, and small in size.

In our previous study, Field Programmable Gate Arrays (FPGAs) have been used as the embedded system to implement our designed NMI with Linear Discriminant Analysis (LDA)-based classifiers [3]. The prototype demonstrated promising performance for real-time NMI implementation. Although extremely effective, FPGAs pose many challenges during the design stage, such as language syntax, design environment, and toolsets [4]. Another concern with the use of FPGAs is its requirement of special purpose hardware design and fabrication giving rise to high cost. For example, a Support Vector Machine (SVM)-based classifier improved the accuracy of NMI for intent recognition compared to LDA [1]. However, hardware programming the complex SVM algorithm on a FPGA is challenging and time

consuming. These difficulties limit our capability to further optimize and develop the NMI for neural control of powered lower limb prostheses.

With the wide availability of commodity off-the-shelf hardware such as Personal Computers (PCs), an efficient and cost-effective way of implementing our NMI is to develop an NMI program specifically tailored to such Commercial of the Shelf (COTS) hardware. Existing PC implementations of our SVM-based NMI algorithms, however, are mainly based on MATLAB giving rise to high overheads and poor real-time performance. Our objective here is to develop a C program realizing our NMI algorithm on a commodity PC that is portable and fast enough.

One alternative to FPGA and regular CPU is a mobile CPU. Mobile CPUs are low cost, low power and much smaller devices than regular CPUs (as shown in Fig. 1.1 [5]). In addition, they have the capability to provide the flexible design environment as a PC/CPU combination. However, the computational power of mobile CPUs, such as the Intel Atom™ Z530, is relatively low [6, 7]. Therefore, in this study, we are interested to investigate whether or not a mobile CPU can execute a highly

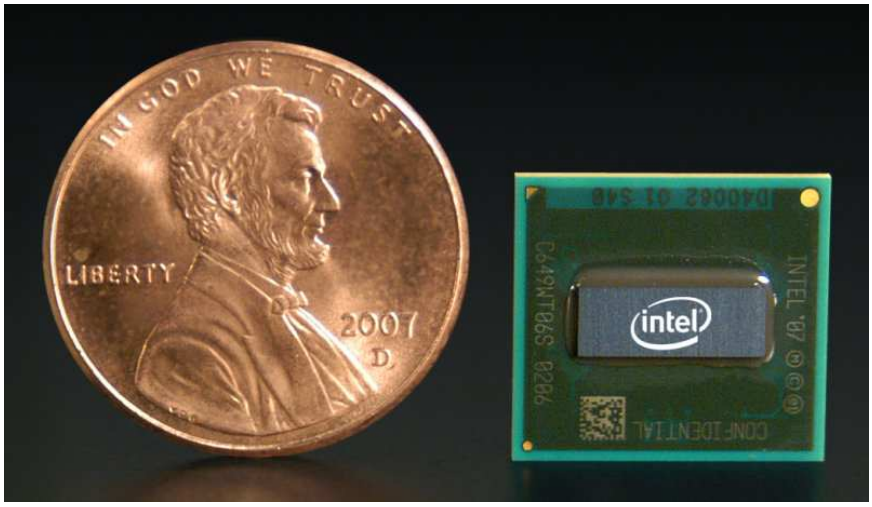

Figure 1.1. Intel Atom™ mobile CPU size compared to a United States penny (a United States penny is approximately 19.05 millimeters in diameter)

computational intensive algorithm, such as our phase-dependent, SVM-based NMI for powered lower limb prostheses.

This paper makes the following contributions:

- Design and implementation of a NMI for artificial legs based on mobile processors;
- Design and implementation of a highly optimized, C-based, embedded application tailored to execute a phase-dependent NMI with SVM classifiers;
- A performance analysis that evaluates the potential of mobile processors for embedded implementation of a NMI for neural control of powered lower limb prosthesis.

1.2 System Design

1.2.1 Hardware Architecture

To provide viable use capability of a NMI, the NMI must be small, dissipate low power, and be fast enough to execute the classification algorithm in real-time. To meet these requirements, the AxiomTek eBOX530-820-FL fanless embedded hardware with the Intel Atom™ Processor Z530 (512K cache, 1.6 GHz) was chosen [8]. The Intel Atom™ Processor Z530 provided the highest performance and lowest power dissipation of Hyper-Threading capable mobile CPUs, which is ideal for thermally constrained and fanless embedded applications [9, 10]. The Hyper-Threading technology allows the operating system and the NMI application to execute simultaneously on two Hyper-Threads as they would on two physical processors [11].

This minimizes the impacts of the OS execution on the real time embedded NMI application.

1.2.2 Software Architecture

C was chosen as the software language in our study because of its superior performance for real-time embedded applications [12-15]. To enhance the system performance, several programming techniques were used in the design and implementation of the application. First, dynamic memory management is one of the most expensive operations in C applications [16], which may cost 30% of the total execution time for the heap intensive C applications [16]. To avoid this problem, the various data structures within the software were defined statically with pre-defined maximum sizes. Secondly, to increase the reliability of the application, the data structures were placed in the application's data segment, not in the application's stack [17], to help avoid stack overflows. Other performance enhancements implemented included loop unwinding [18] and inline function expansion [19]. Loop unwinding is an efficient means to increase the utilization of pipelines and helps eliminate loop overhead [18]. Inline function expansion replaces a function call with the body of the function, which reduces the overhead associated with a function call during program execution [19].

The designed Neuromuscular-Mechanical fusion PR algorithm, utilizes SVM classification. The open source library LIBSVM [20] was used and specifically tailored to our embedded NMI application for real-time SVM classification. LIBSVM was also utilized in our previous MATLAB implementation, which served as a baseline for accuracy determination of the embedded application.

1.3 Pattern Recognition Algorithm

The previously developed NMI identifies the user's locomotion mode based on electromyographic (EMG) signals recorded from the residual thigh muscles and mechanical forces/moments signals recorded from prosthetic pylon. These EMG and mechanical data are segmented by the sliding analysis windows. Features are extracted from the raw EMG and mechanical data in each analysis window and fused into one feature vector. This feature vector is sent to a phase-dependent pattern classifier for determination of user intent. The phase-dependent pattern classifier consists of multiple sub-classifiers for individual defined gait phases and a gait phase detector that identifies current gait phase and switches the corresponding sub-classifier on. Detailed description of this previously designed NMI can be found in [1] and [2].

1.3.1 Feature Extraction

In this study, four time-domain (TD) features (the mean absolute value, the number of zero crossings, the waveform length, and the number of slope sign changes) were extracted from EMG signals in each analysis window. For mechanical measurements, the mean, minimum, and maximum values in each analysis window were extracted as the features. More detailed information can be found in [1]. The length of sliding analysis window and window increment were 150ms and 50ms, respectively.

The features and increments were chosen to match our previous MATLAB implementations [21], thereby providing a baseline for an accuracy comparison with the newly designed embedded application.

1.3.2 Phase Dependent Pattern Recognition

To accurately determine user intent, SVM utilizing a Radial Basis Function (RBF) kernel [21] was utilized. The SVM gamma parameter of 0.015 was used.

In the designed phase-dependent classifier, four sub-classifiers were defined corresponding to the following four gait phases: initial double limb stance (phase 1), single limb stance (phase 2), terminal double limb stance (phase 3), and swing (phase 4) [21]. The gait phase detector detects these gait phases based on the vertical Ground Reaction Force (GRF). In order to build the parameters in the classifiers, training procedure must be conducted on a training data set. During training, the output of phase detector is used to label the training data with the corresponding gait phase. Each classifier is trained only with the data pertinent for its gait phase. When testing the classification, the gait phase detector determines which classifier is responsible for the determination of user intent. The algorithmic data flow of the phase-dependent pattern recognition is shown in Fig. 1.2.

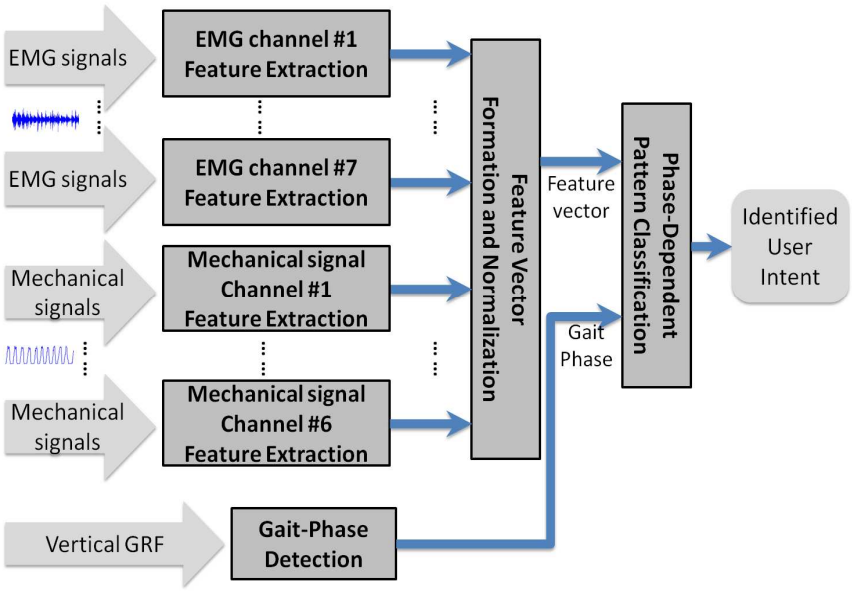

Figure 1.2. Phase-dependent PR algorithmic data flow

1.3.3 Software Implementation

To implement the Neuromuscular-Mechanical Fusion PR, three applications were developed. The first application accepts offline raw training data, performs the EMG and mechanical feature extraction, fuses and then normalizes the features into vectors. The feature vectors are then separated into their corresponding gait phases and provided to the training application. The first application is also responsible for generating the normalization parameters required by the PR to normalize the testing data, when determining user intent. The second application accepts the four sets of training vectors and generates four SVM models, one model for each gait phase. The third application accepts raw offline testing data, the four gait phase SVM models, and the normalization parameters. The application extracts EMG and mechanical features from the raw testing data. The features are then fused and normalized, with the provided normalization parameters, into a vector. Finally, the application determines the current gait phase, and forwards the test vector to the respective phase based classifier for determination of user intent. The software implementation data flow is shown in Fig. 1.3.

1.4 Performance Evaluation

This study was conducted with approval of Institutional Review Board (IRB) at the University of Rhode Island and informed consent of the subject. The evaluation was performed offline on the data collected from a male subject with a transfemoral amputation. The collected data included the EMG signals from the subject's residual thigh muscles and mechanical forces/moments measured by a 6 degree-of-freedom load cell mounted on the prosthetic pylon. The monitored residual muscles included

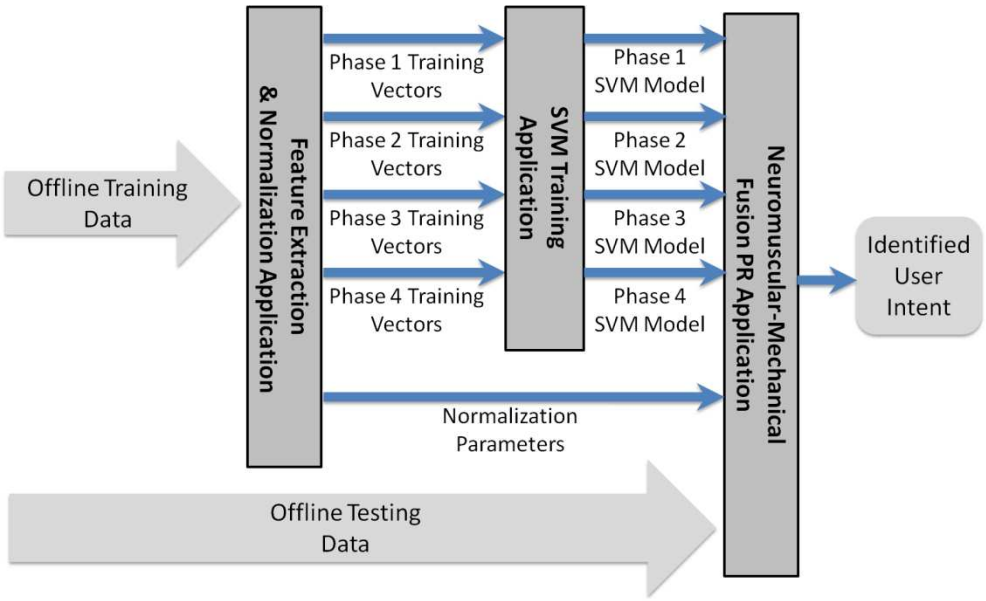

Figure 1.3. Software implementation data flow

the rectus femoris (RF), vastus lateralis (VL), vastus medialis (VM), biceps femoris long head (BFL), semitendinosus (SEM), biceps femoris short head (BFS), and adductor magnus (ADM). The recognition accuracy of NMI by using the designed embedded system was compared with the results of existing PC implementations on MATLAB. In addition, the timing and processor loading of the application's execution on the embedded hardware were evaluated. A power consumption comparison between similar proposed NMI embedded systems and this embedded system was provided.

1.4.1 Recognition Accuracy of NMI

The offline data was composed of seven different classes: level-ground walking, ramp ascent, ramp descent, stair ascent, stair descent, sitting, and standing. The comparison of recognition accuracies of the NMI by using the designed embedded system and existing PC implementations on MATLAB are provided in Table 1.1. This study utilized a slightly different value for the gamma parameter required by the SVM

Table 1.1. MATLAB and embedded software classification accuracies

CLASSIFIER	MATLAB MODEL	EMBEDDED SW
PHASE 1	97.74%	98.33%
PHASE 2	96.72%	98.82%
PHASE 3	98.63%	98.67%
PHASE 4	95.18%	95.66%

classifiers. The different gamma value was shown to provide a slightly higher accuracy during testing. This is noticeable in the comparison results, whereby the embedded application slightly outperformed the MATLAB model in PR accuracies.

Both the MATLAB results and the embedded application had lower Phase 4 (swing) accuracies. Two explanations for this result are provided in [22]. The first is that there is little force/moment data present during the swing phase from the prosthetic pylon [22]. The second explanation is related to the swing phase being longer than any of the other three phases, leading to larger variations in the EMG features [22].

1.4.2 Execution Timing and Processor Loading on the Embedded Hardware

This previously designed NMI algorithm was executed on the Intel Atom™ based embedded hardware and the performance results were evaluated. A total of 3555 predictions were produced by the Intel Atom™ based embedded hardware. For the purpose of this evaluation, the prediction time will be defined as the total time to execute feature extraction, normalization, gait phase detection and classification for a single analysis window. The mean prediction time was 0.8455 milliseconds with a standard deviation of 0.1044 milliseconds. The worst case prediction executed in 2.1265 milliseconds. These results clearly show that the embedded system is capable

of real-time implementation at 50ms and 20ms window increments. If the embedded system is combined with a highly responsive Data Acquisition (DAQ) system to provide the EMG and mechanical data, even a window increment of 10ms may be feasible. At the 10ms window increment, the interface to the DAQ and the DAQ system drivers will become of the utmost importance.

Because there is additional loading on the CPU to execute the data logging for post analysis, the CPU loading provided by the operating system may be inaccurate. Therefore the mean and maximum value of CPU loading was calculated by (1.1) which were 1.691% and 4.253% respectively.

$$CPU\ Loading = \frac{Prediction\ Time}{Window\ Increment\ (50ms)} * 100 \quad (1.1)$$

1.4.3 Power Consumption Comparison

Previous studies have utilized Field Programmable Gate Arrays (FPGA) and PCs for similar NMI applications [23]. The reported power consumption for the FPGA was 3.499 watts and the AMD Turion 64x2 CPU within [23] can utilize up to 35 watts [23]. The Intel Atom™ Z530 Processor utilized in this embedded system design dissipates 2.2 watts [9]. The Intel Atom™ CPU's power dissipation is less than one-fifteenth that of the CPU and less than two third that of the FPGA.

1.5 Conclusions

This paper presented the design and implementation of a mobile CPU based embedded system for a NMI for artificial leg control. The performance evaluation showed that the highly optimized C-based embedded application combined with the mobile-CPU-based embedded hardware, can easily meet real-time constraints. The

performance evaluation also shows that there is no loss in classification accuracy, when compared with the MATLAB model [21]. In fact, there is a slight increase due to the use of a different SVM gamma parameter. Lastly, the CPU utilized for this embedded system dissipated less power than other systems designed for similar applications. Future work to be performed includes interfacing the embedded system to a DAQ to create a real-time capable system and testing the system on lower limb amputees.

List of References

[1] H. Huang, F. Zhang, L. J. Hargrove, Z. Dou, D. R. Rogers, and K. B. Englehart, "Continuous locomotion-mode identification for prosthetic legs based on neuromuscular-mechanical fusion," IEEE Trans Biomed Eng, vol 58, pp. 2867-75, 2011.

[2] H. Huang, T. A. Kuiken, and R. D. Lipshutz, "A strategy for identifying locomotion mode using surface electromyography," IEEE Trans Biomed Eng, vol 56, pp. 67-73, 2009.

[3] X. Zhang, Q. Yang and H. Huang, "A Neural-Controlled Cyber Physical System for Intent Recognition for Artificial Legs," presented at Design Automation Conference, San Francisco, 2012 (Accepted).

[4] I. Gonzalez, E. El-Araby, P. Saha, T. El-Ghazawi, H. Simmler, S. Merchant, B. Holland, C. Reardon, A. George, H. Lam, G. Stitt, N. Alam, M. Smith, "Classification of application development for FPGA-based systems," Conf Proc National Aerospace Electronics Conference, 2008

[5] J. Mahoney. (2008, July) "Intel CEO: Atom Platform Something 'Most of Us Wouldn't Use'" [online] Available: http://gizmodo.com/5026401/intel-ceo-atom-platform-something-most-of-us-wouldnt-use [March 22, 2012]

[6] PassMark Software. (2012, March) "PassMark CPU Benchmarks - Low End CPU's," [online] Avaliable: http://www.cpubenchmark.net/low_end_cpus.html [March 22, 2012]

[7] B. Crothers. (2008, April) "New Intel design may spur (more) tiny PCs," [online] Avaliable: http://news.cnet.com/8301-13924_3-9928126-64.html [March 22, 2012

[8] AxiomTek Corporation. (2012). “Fanless Embedded System with Intel® Atom™ Processor” [online]. Available: http://axiomtek.com/Download/Spec/ebox530-820-fl.pdf [March 19, 2012]

[9] Intel Corporation. (2010, June). “Intel® Atom™ Processor Z5xx Series Datasheet” [online]. Available: http://www.intel.com/content/www/us/en/processors/atom/atom-z540-z530-z520-z510-z500-45-nm-technology-datasheet.html [March 19, 2012]

[10] Intel Corporation. (2011, April). “Intel® Atom™ Processor Z6xx Series Datasheet” [online]. Available: http://www.intel.com/content/www/us/en/processors/atom/atom-z6xx-datasheet.html [March 19, 2012]

[11] D. Marr, F. Binns, D. Hill, G. Hinton, D. Kaoufaty, J. Moller, M. Upton “Hyper-Threading technology architecture and microarchitecture,” Intel Technology Journal, vol. 6 no. 1, pp 4-15, 2002

[12] M. Weiskirchne. (2003, September). Comparison of execution times of Ada, C and Java. EADS Deutschland GmbH Military Aircraft. Muenchen, Germany. [online] Available: http://www.aicas.com/info/EADS_benchmark_languare_comparison.pdf [March 19, 2012]

[13] P. Sestoft. (2010, February). Numeric performance in C, C# and Java. University of Copenhagen. Copenhagen, Denmark. [online] Available: http://www.itu.dk/~sestoft/papers/numericperformance.pdf [March 19, 2012]

[14] Cherrystone Software Labs. (2010, August). Algorithmic performance comparison between C, C++, Java and C# programming languages. Cherrystone Software Labs. Boston, MA [online] Available: http://www.cherrystonesoftware.com/doc/AlgorithmicPerformance.pdf [March 19, 2012]

[15] P. Ambika, H. Ahmed. (2001). A performance analysis of Java and C, University of Columbia, New York, NY, Available: http://www.cs.columbia.edu/~sedwards/classes/2001/w4995-02/litsurveys/haseeb-ambika.pdf [March 19, 2012]

[16] D. Tiwari, S. Lee, J. Tuck, Y. Solihin. “Exploiting fine-grained parallelism in dynamic memory management,” IPDPS, 2010

[17] L. Ferres. (2010). Memory management in C: The heap and the stack. Universidad de Concepcion, Concepcion, Chile. [online] Available: http://www.inf.udec.cl/~leo/teoX.pdf [March 19, 2012]

[18] A. Nicolau, “Loop quantization: unwinding for fine-grain parallelism exploitation,” Cornell Univerisy, 1985, Available:

http://ecommons.library.cornell.edu/bitstream/1813/6549/1/85-709.pdf [March 19, 2012]

[19] W. W. Hwu, P. P. Chang, “Inline function expansion for compiling C programs,” ACM SIGPLAN ’89 Conference on Programming Language Design and Implementation, Portland, Oregon, June 1989

[20] C. C. Chang, C. J. Lin, “LIBSVM: a library for support vector machines,” ACM Transactions on Intelligent Systems and Technology, vol. 2 issue 3, pp. 27:1-27:27, 2011

[21] F. Zhang, H. Huang, “Real-Time Recognition Of User Intent For Neural Control Of Artificial Legs,”MEC’11, New Brunswick, Fredericton, NB Canada, August 2011.

[22] H. Huang, F. Zhang, L. J. Hargrove, Z. Dou, D. Rogers, and K. B. Englehart, “Continuous locomotion-mode identification for prosthetic legs based on neuromuscular-fusion,” IEEE Trans Biomed Eng, vol. 58 issue 10, pp. 2867-2875, October 2011

[23] X. Zhang, H. Huang, Q. Yang, “Design and implementation of a special purpose embedded system for neural machine interface," Conf Proc IEEE International Conference on Computer Design, 2010, pp. 166-72.

MANUSCRIPT 2

Towards Policy and Guidelines for the Selection of Computational Engines

by

[1] Robert Hernandez, John Faella

is published in *the proceedings of the 7th Annual*

IEEE Systems Conference (SysCon '13),

Orlando, FL, 2013. p. 88-95.

[1] Robert Hernandez and John Faella are with Torpedo Systems Department, Naval Undersea Warfare Center, Newport, RI, 02842, Email { robert.hernandez2, john.faella }@navy.mil.

Abstract

Much research has been performed that concentrates on providing processing throughput enhancements to existing algorithms. Many systems have performance requirements that constrain their volume and/or power consumption. For volume and power consumption constrained systems, throughput cannot be the only decision factor when selecting a computational engine. Typical studies can aid in the selection of computational engines that meet the throughput requirements of a system, but may be of little help with respect to the volume, power and thermal constraints. This paper takes a different approach to help provide a different perspective on the constrained design problem. The research performed in this paper emphasizes the cost due to the power, size and Non-Recurring Engineering (NRE) costs of various computational engines. The computational engines researched in this paper are: Central Processing Unit (CPU), mobile CPU, Digital Signal Processor (DSP), and mobile Graphics Processing Unit (GPU). The various architectures are compared against each other with respect to throughput, power, size and NRE costs. The authors hope that the process outlined in this paper may serve as a possible guideline for other Systems Engineers to perform similar Analysis of Alternatives of computational engines. Furthermore, the authors hope that the methods used for the relative performance evaluations will serve as a starting point to help shape policy in the selection of computational engines for future designs.

2.1 Introduction

When performing an Analysis of Alternatives (AoA) for the selection of the computational engine of a system, attention needs to be paid to the system constraints. Much research has been performed that concentrates on providing processing throughput enhancements to existing algorithms [1, 2, 3, 4], but many systems have performance requirements that constrain their volume and/or power consumption. Studies such as [1, 2, 3, 4] can aid in the selection of computational engines that meet the throughput requirements of a system, but may be of little help with respect to the volume, power and thermal constraints. If the limitations of the chosen architecture are not well understood beforehand, the results can be expensive and time consuming. Furthermore, if the benefits of each computational engine are not well understood beforehand, an inferior or inappropriate architecture may be chosen. This results in reduced system capability, thereby limiting the current and future software algorithms that can be implemented. Therefore, it is important to understand the limitations and benefits of existing hardware architectures and provide the best system design alternatives based on each system's specific performance requirements and constraints. This research is a direct result of this initiative and provides a methodology for performing AoAs of existing computer architectures for use in future Naval Systems. The intent is that this research may serve as guidelines and enable system engineers to choose the most appropriate architecture for use in their particular system. The primary focus will be providing guidelines for systems that are constrained, such as volume constrained, power constrained, or both power and volume constrained. The guidelines will be useful for system engineers whose

applications are unconstrained, but the primary focus of this paper will be the constrained design analysis. The viable architectures analyzed in this study are: Central Processing Unit (CPU), mobile CPU, Digital Signal Processor (DSP), and mobile Graphics Processing Unit (GPU).

To help systems engineers and designers choose the appropriate architectures, this study provides the following contributions:

- Data on the software development Non-Recurring Engineering cost (NRE) for the DSP and GPU architectures for porting from a C-based application to aid in producing accurate NRE estimates and schedules;
- Architecture based performance assessments related to power utilization, space utilization and SWaP (space, wattage and performance) [5] to aid in meeting system performance requirements and constraints;
- Architecture specific overhead, such as GPU Kernel function overhead, to better understand the complexity and limitations of the architectures.

A candidate algorithm has been chosen that performs signal processing on multiple raw data streams and utilizes Support Vector Machine (SVM) based classification [6, 7]. The candidate algorithm was chosen for its similarity with processing requirements for many naval systems as well as the research's applicability to the Wounded Warrior Program. This particular algorithm is a Neural Machine Interface (NMI) for volitional control of powered lower limb prostheses. A NMI application is both volume and power constrained, but also requires a significant amount of processing throughput, which poses many challenges [8]. To develop the candidate algorithm, the MATLAB model utilized in [6] and [7] was ported to an

ANSI C baseline and its accuracy verified against the MATLAB model. The first candidate architecture to undergo a performance evaluation was the mobile CPU, because of its direct applicability to the NMI's constraints (i.e. - high performance utilizing a small and low power device). The performance results for the mobile CPU based NMI were published in [8]. This paper provides the additional performance results for a CPU, DSP, and mobile GPU. Furthermore, it provides an architecture performance comparison of all four architectures, thereby providing the basis of an AoA for the selection of hardware architectures.

The paper is organized as follows. The next section presents the Neural Machine Interface Algorithm. Sections III, IV and V present our implementation and performance for the various architectures (i.e. - computational engines). Sections VI, VII and VIII provide our constrained performance evaluations. We conclude our paper in Section IX.

2.2 Neural-Machine Interface

This NMI utilizes a pattern recognition (PR) algorithm that identifies user locomotion intent based on seven (7) electromyographic (EMG) signals acquired from leg muscles and six (6) mechanical forces/moments data acquired from a 6 degrees-of-freedom (DOF) load cell mounted on the prosthetic device. Time domain based features are extracted from this data and provided to SVM-based gait phase classifiers for determination of user intent. A brief description of the NMI PR algorithm is provided below, a detailed description is available in [6] and [7].

2.2.1 Support Vector Machine Classification

SVM is a supervised learning classification technique whereby the selection of the features utilized for training and detection directly relate to classification accuracy and burden placed on the computational engine [9]. SVM supports the use of non-linear kernel functions [9], such as the Radial Basis Function (RBF), which provides the capability to better match the distribution of the feature sets. The chosen algorithm utilizes SVM with an RBF kernel function to provide its user intent classification. The features were chosen to provide high accuracy and minimize the burden on the computational engine [10].

2.2.2 Feature Extraction

In this study, four time-domain (TD) features (the mean absolute value, the number of zero crossings, the waveform length, and the number of slope sign changes) were extracted from EMG signals in each analysis window [10]. For mechanical data the mean, minimum, and maximum values in each analysis window were extracted as the features.

2.2.3 Phase Dependent Pattern Recognition

The user's human locomotion is separated into four gait phases: initial double limb stance, single limb stance, terminal double limb stance, and swing [11]. Four separate detectors are trained, each with the data from a single corresponding gait phase. Data features are extracted from the raw EMG and mechanical signals during a sliding analysis window and fused into a single feature vector. A gait phase detector identifies the current gait phase in real-time, selects the corresponding gait sub-classifier, and forwards the feature vector to the classifier for final determination of

user intent. In this study, a sliding analysis window of 150ms with a window increment of 50ms was utilized.

2.2.4 Performance Evaluation of the NMI

The performance evaluation of the NMI on the various architectures will be directly related to the average prediction achieved by the architectures. For the purposes of the various evaluations, the prediction time will be defined as the total time to execute: feature extraction, normalization, gait phase detection and classification for a single analysis window.

2.3 CPU and Mobile CPU Implementation and Performance

The CPU and mobile CPU implementations were directly based on the C language implementation of the research performed in [8]. In [8], the goal was to create a NMI capable of meeting real-time constraints, while executing on low power architectures. To help the lower power architectures meet real-time constraints, various common performance enhancements techniques were implemented. These enhancements included reduced dynamic memory management [12], loop unwinding [13], and inline function expansion [14] among others. The NMI's average prediction time, during execution on an Intel Atom Z530, was 0.846ms. The Intel Atom Z530 CPU has a form factor of 13mm x 14mm and has a maximum power utilization of 2.2 watts [15].

The current NMI CPU implementation was written to take advantage of single core hyper threaded [16] CPUs and is, therefore not capable of taking full advantage of multi-core CPU architectures such as Intel's i5 and i7 CPUs. The closest CPU comparison to the execution on the Atom Z530 we had available was the Intel E7500

Core 2 Duo. Similarly to the previous study, the Intel E7500 allowed the Operating System (OS) to execute on one core, while the NMI executes on the second core. This helps minimize the impacts of the OS on the NMI. The NMI's average prediction time during execution on an Intel E7500, was 0.605ms. The Intel E7500 CPU has a form factor of 37.5mm x 37.5mm and has a maximum power utilization of 65 watts [17].

2.4 DSP Implementation and Performance

The DSP implementation began with the mobile CPU C software baseline. The C baseline was modified and optimized to work with the Spectrum Digital TMS3206713 board that utilizes a Texas Instruments TMS3206713 DSP [18] at a clock speed of 225MHz. The development board was programmed in the C programming language using the provided Code Composer Studio integrated development environment.

For a professional with prior C programming experience, but no prior experience using Code Composer Studio, it took about 1 week to get a non-optimized program to match the mobile CPU version's execution time and accuracy. An additional 2 weeks of time was required to optimize the application to reach its maximum potential.

One optimization performed was to reduce the number of branches required by the program. The TMS320C6713 does not have any form of branch prediction. Instead, each branch function results in 5 stall operations being inserted into the pipeline [19]. When possible, the instances of nested if statements were merged into a single if statement, thus reducing the number of branches required for the same operation. The number of conditional loops was reduced by combining multiple operations into a single loop whenever possible. This also reduced the number of branches that occur within the program.

The most effective optimization was the activation of L2 cache. The TMS3206713 development board does not have L2 cache activated by default [20]. Instead only a small L1 cache is used. Since the external memory accesses are slow, it is beneficial to activate the L2 cache as long as the application execution does not result in a large number of cache misses. The inclusion of the L2 also requires the remapping of some internal memory to be configured to serve as the cache. In this case neither of these two issues were a factor and the inclusion of L2 cache provided a major performance boost. This change required an additional two lines of code to be added to the program. The first instruction configures the board to use L2 cache, and the second instruction can be used to control the size of the L2 cache. In this case it was found that the largest performance was achieved when with the largest possible L2 cache. For the TMS320C6713 development board the largest possible L2 cache size is 64KB [20].

The optimized version of the DSP implementation resulted in an average time of 11.35ms per prediction with a standard deviation of 2.186ms. The feature extraction required an average of 6.887ms with a standard deviation of 838µs. The classification required an average of 4.472ms with a standard deviation of 1.778ms. The TMS3206713 DSP has a form factor of 27mm x 27mm and has a power utilization of approximately 1 watt [18].

2.5 Mobile GPU Implementation and Performance

The mobile GPU implementation began with the mobile CPU software baseline. The C baseline was modified and optimized to take advantage of the Nvidia GeForce GT 540m architecture. The GPU utilized in this study is located within a Dell XPS

laptop with an Intel i7-2720QM CPU. The GeForce GT 540m has 96 Compute Unified Device Architecture (CUDA) cores divided up into 2 separate streaming multiprocessors and runs at a clock speed of 1.3GHz [21]. The development of the application was performed in a Microsoft Visual Studio integrated development environment which provides CUDA programming capability.

One immediate difference in the GPU architecture versus the DSP and CPU architectures is that the GPU is more of a highly optimized and parallelized co-processor to the CPU than it is a standalone architecture. Therefore, the GPU incurs the additional power and space overhead of the CPU or device it communicates with. Because the CPU power and form factor can vary, our analyses will not take into consideration this additional overhead. It is recommended that this implementation specific overhead be accounted for by the systems engineer, while performing the analyses within this paper. Another disadvantage of the GPU is the time of the overhead required to launch a GPU kernel function. The CPU needs to communicate with the GPU in order to setup and run a CUDA kernel function. There is a certain amount of overhead time required to perform this process. If the kernel launch overhead begins to approach or exceeds the actual execution time of the kernel function then it can become a detriment to the total execution time of the program. Therefore, if one has a kernel function that performs little to no calculations, attention needs to be paid to how time is spent in actual kernel function execution versus the kernel launch overhead. In some cases it may be more advantageous to execute the less calculation intensive functions on the CPU, thereby eliminating the need for kernel function overhead. In the case of our implementation of the gait phase detection

and tallying of SVM vote, it was beneficial to execute these on the CPU versus the GPU. For this system, it was found that an average of 3µs of overhead time is required per GPU kernel launch. Our GPU implementation utilized nine (9) GPU kernel functions per prediction; therefore a total of 27µs per prediction is attributed to kernel function overhead.

One important concept in GPU programming is the concept of organizing execution paths into grids, blocks, threads, and warps. When starting a kernel function the CPU specifies several parameters. The main parameters used are the number of threads per block, the number of blocks, and the number of grids of blocks. In this case there was only one grid, since we were using a single GPU board. Threads are grouped into blocks. Threads in the same block can share data and be synchronized whereas threads of different blocks cannot. Another important concept is warps. Threads are grouped into sets of 32 threads known as warps. Threads in the same warp are intrinsically synchronized and are scheduled together. When writing GPU code it is important to keep threads of the same warp following the same execution path to prevent divergent warps. When threads of the same warp execute different code the warp is said to be divergent and the operations are executed in a serialized fashion, thus missing the potential parallelism offered by the GPU. More detailed information on CUDA programming, grids, blocks, threads and warps can be found in [22].

Our CUDA program begins its execution on the CPU and then the CPU initiates kernel functions that execute on the GPU. In this case the program begins by copying the raw EMG and load cell data to the GPU to be used during its execution. For each

analysis window, the phase detection is performed on the CPU. Nine GPU kernel functions are used to perform the needed steps for the feature extraction, normalization, and to determine the one versus one SVM classifier votes. The votes are then copied from the GPU to the CPU where the actual one versus one SVM votes are tallied to determine the user intent for the given window.

This NMI algorithm allows for a large amount of parallelization. The GPU's massively parallel architecture provides the capability to take advantage of this opportunity. As shown by Amdahl's Law [23], the more parallelization that can be found in an application, the greater the increase in the performance of the application on a parallel architecture such as a GPU. To take advantage of the principles defined by Amdahl's Law we examined the NMI algorithm for every possible opportunity to exploit parallelism. The DC offset for each of the channels can be calculated and removed in parallel. Each of the 46 features that need to be extracted from the channel data can be calculated in parallel. Similarly, the approximately 200 to 400 SVM support vector dot products can be performed in parallel, and the 21 one versus one SVM classifiers that use the SVM dot product values can be performed in parallel.

The parallelization was further increased by utilizing the parallel reduction method [24, 25] to parallelize the necessary work to calculate the values. The parallel reduction method uses many threads to process a data set. For example, when finding the sum of a data set each thread will be used to find the sum of two values in the data set. After the first stage, half of the threads will have partial sums. Then half of the threads with the partial sums add their resultant partial sums to that of one of the other threads. This process continues until one thread holds the sum of the entire data set.

In this way the sum is found in the most efficient way to maximize the parallelism provided by the GPU [24, 25].

Fig. 2.1 shows the NMI algorithm's GPU implementation, data flow, and the workload separation between the CPU and GPU architectures. In Fig. 2.1, the portions of the algorithm allocated to the GPU are initiated by kernel functions launched by the CPU to perform the calculations. These kernel functions are launched with a set amount of blocks and threads per block in order to best take advantage of the architecture of this particular GPU. For our program we utilized a block size of 96 threads. This provided enough threads to accomplish each given task.

For the six EMG channels, the DC offset first has to be removed. This is done by calculating the mean of each channel and subtracting the mean from each of the

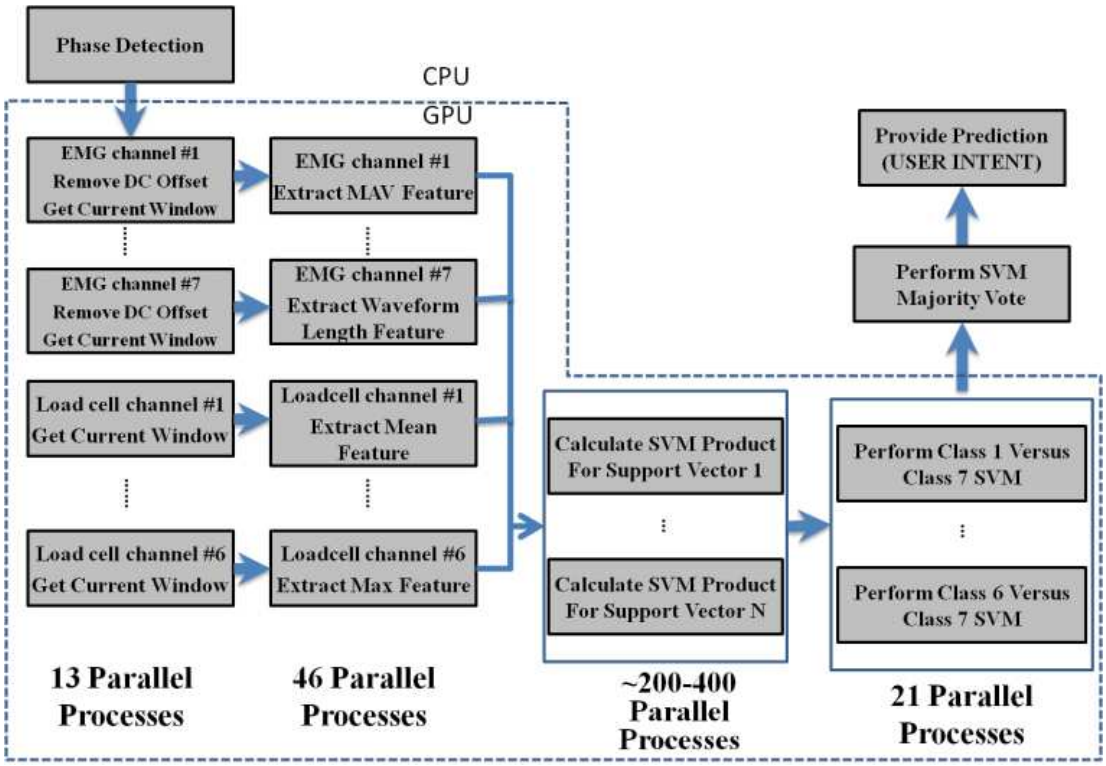

Figure 2.1. Phase-dependent PR algorithmic data flow implementation in a GPU architecture

values. The mean of each channel is calculated in the first GPU kernel function. This function finds the means for all channels including the mechanical channels as this is a feature required from the mechanical channels. A second kernel function is then used to subtract the mean from the EMG channels.

For the feature extraction, 96 threads per block are used to calculate the features needed for a channel. Each block is responsible for extracting the features for 1 channel. Therefore 13 blocks are required, seven for the EMG channels and six for the mechanical features. All the features are calculated using the reduction method. The third kernel function is used for the extraction of the EMG features. In this function each thread reads in a single data value from a single EMG channel and then extracts all four EMG features for this data point (calculates the absolute value of the current data point, calculates the waveform length of the current data point relative to the prior data point, determines if the current data point is representative of a zero crossing, and determines if the current data point is representative of a slope sign change). Once these factors are known, the values can then be combined together using parallel reduction as previously described until one thread holds the feature values for the current window increment. The mechanical channels do not have to wait for the DC offset to be subtracted from the data so the features from these channels can begin to be extracted immediately while the EMG channels are still waiting. There is no need for a separate kernel function to calculate the mean of the mechanical channels as that is handled by the same kernel function that calculates the mean of each channel in order to remove the DC offset from each of the EMG channels. The fourth kernel function is used to extract the mechanical features and

also utilizes the reduction method to increase parallelism. Each thread reads in one value from the current window. It then compares this value to the value in one other thread to determine which is the minimum and which is the maximum. This process continues until there are only two threads remaining at which point the min and max for the entire data set can be found. The fifth kernel function saves the features by loading them into an array to be processed by the later steps in the GPU implementation of the algorithm.

A sixth kernel function is utilized to calculate the SVM dot products and also utilizes the reduction method. Each block is allocated 96 threads. Each block of 96 threads is segmented into three warps of 32 threads. Each warp calculated the value for one SVM dot product; hence the three warps can calculate 3 SVM dot products in parallel. There are a total of 46 features in each SVM support vector and test vector, therefore 46 products and 45 sums are required for each dot product. Each warp performs the following steps:

Step 1. The 32 threads in the warp calculate the first 32 products.

Step 2. The first 14 threads in the warp calculate the final 14 products and sums them with their prior 14 products, resulting in the first 14 partial sums.

Step 3. The 32 terms (18 remaining products and 14 partial sums) are reduced into 16 partial sums.

Step 4. The remaining 16 partial sums are reduced into 8 partial sums.

Step 5. The remaining 8 partial sums are reduced into 4 partial sums.

Step 6. The remaining 4 partial sums are reduced into 2 partial sums.

Step 7. The remaining 2 partial sums are summed and become the final sum.

We allocated a single SVM dot product to a warp of 32 threads to minimize warp divergence and synchronization issues. If threads from the same warp follow different execution paths then the threads are said to diverge. In the case that the threads diverge, they are executed in a serial fashion and thus do not take best advantage of the parallel processing provided by the GPU. The other advantage of using a warp to calculate a single SVM dot product is that there is no need to call any thread synchronization functions because the threads of a warp are naturally synchronized.

An eighth kernel function is executed at the same time as the SVM dot products are being calculated. This function does some necessary setup prior to the SVM classification. In this function some required variables are initialized to be used in the classification. The ninth kernel function performs the one versus one SVM classification. The 21 classifiers are executed using 21 blocks of 96 threads each. Each of the classifications is again done using parallel reduction. This produces the 21 votes that are copied back to the CPU in order to tally the final vote and determine the user intent for the current window.

This implementation takes advantage of the parallel nature of the GPU while at the same time avoiding one of its biggest disadvantages, the need to copy data back and forth between the GPU and the CPU [22]. With the method outlined above the program only requires one memory copy between the GPU and the CPU per prediction. This is done by keeping as much of the data as possible on the GPU and

only copying data to the CPU at the very end of a prediction. Fig. 2.2 shows the GPU program flow for this implementation.

About a month and a half of work performed by a professional with prior C experience and no prior CUDA experience was required to produce a GPU implementation that matched the accuracy of the CPU implementation. An additional month and a half was required to produce code that could match and exceed the prediction speed accomplished by the CPU implementation. In the final optimized version of the GPU code, the program required an average 0.193ms per prediction with a standard deviation of 21µs. An average of 51µs with a standard deviation of

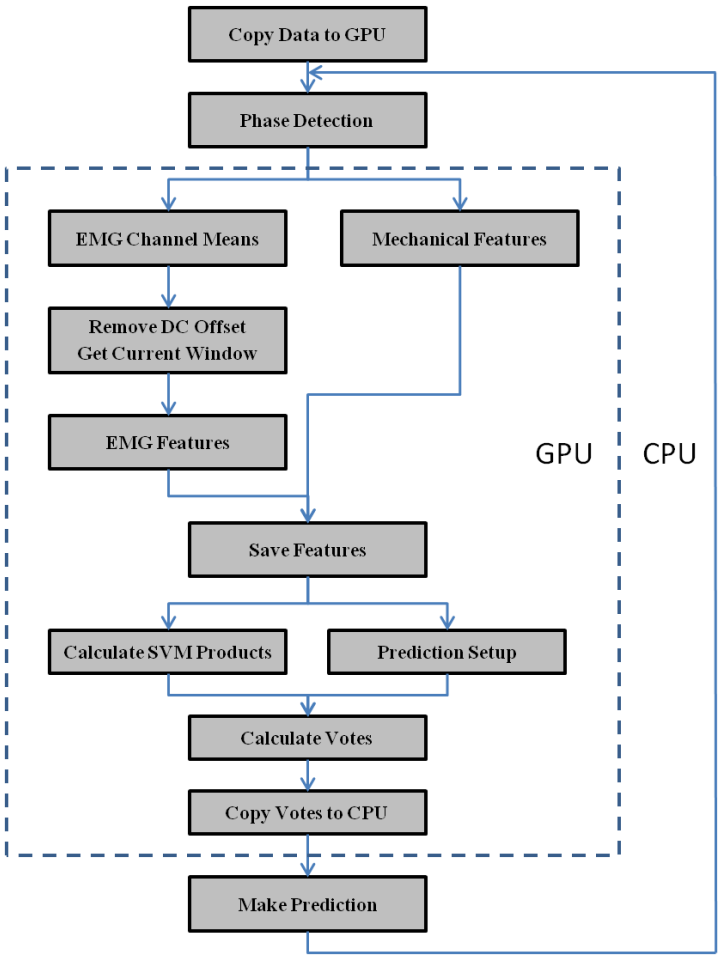

Figure 2.2. GPU implementation program flow

0.2µs was required for the feature extraction and 74µs with a standard deviation of 18µs for the classification. The GeForce GT 540m GPU has a form factor of 29mm x 29mm and has a maximum power utilization of 35 watts [26, 27].

2.6 Mobile GPU Implementation and Performance

This analysis will compare the computational performance of the architectures relative to their respective power utilization. We recommend that this analysis be performed for systems that are constrained to operate within a limited power or thermal range. This performance requirement is usually imposed when the lower power utilization will allow the device to operate for a longer period, there is a limited method to dissipate thermal energy, or the system has a limited power source. Some examples might be satellites, electric passenger vehicles, and electric autonomous vehicles.

For a computational performance measure we will utilize the number of floating point operations per second (FLOPS). The power utilization will be measured in watts. Of interest is the performance of each architecture per its respective power utilization, therefore (2.1) can be used to provide the relative performance of each architecture for our NMI algorithm. We intentionally utilized the same NMI algorithm for all the architectures to ensure that the number of floating point operations per prediction, the numerator in (2.1) below, is the same for all architectures implementations. Therefore to maximize the performance of any given architecture we must minimize the product of the prediction time and power utilization. Conversely, the architecture whose product of the prediction time and power utilization is the largest will be the worst performing architecture for this analysis. For this analysis, the Intel E7500 CPU

architecture exhibited the worst performance with a prediction time of 0.605ms and a maximum power utilization of 65 watts. To provide a comparison between all the architectures we will take a ratio of each architecture's performance achieved by (2.1) relative to the worst performer (CPU). By comparing the architectures to the worst performer we can then provide a performance ratio utilizing (2.2).

$$\mathbf{Perf} = \frac{\mathbf{Floating\ Point\ Operations\ For\ Each\ Prediction}}{(\mathbf{Seconds\ For\ Each\ Prediction}) * (\mathbf{Watts})} \tag{2.1}$$

$$\mathbf{Ratio} = \frac{(\mathbf{CPU\ Prediction\ Time}) * (\mathbf{CPU\ Watts})}{(\mathbf{Architecture\ Prediction\ Time}) * (\mathbf{Architecture\ Watts})} \tag{2.2}$$

Table 2.1 provides the results of the power constrained analysis. As can be seen, the Atom mobile CPU provided the highest performance, which was 21 times that of the CPU. Although the mobile CPU provided the highest performance, it does not automatically make it the best architecture choice; the system performance requirements need to be examined prior to making a final selection. This applies for all

Table 2.1. Power Constrained Performance Results

Computational Engine	Average Prediction Time	Power Consumption	Performance Ratio
CPU Core 2 Duo E7500	0.605ms	65 watts	1X
DSP TMS320C6713	11.35ms	1 watt	3.5X
Mobile GPU GeForce 540m	0.193ms	35 watts	5.8X
Mobile CPU Atom Z530	0.846ms	2.2 watts	21X

of the analyses performed in this paper. For example, for an NMI, the less power utilized the longer a patient is able to use the prosthesis without the need for changing and/or replacing the power cell. All of the architectures tested met the requirement for a 50ms prediction time. Since the DSP's power consumption is lower than that of the mobile CPU, it may be the better choice. Alternatively, since the prosthetic device we are targeting requires updates every 10ms, the goal is for the NMI to perform a prediction every 10ms. Based on a 10ms prediction time, the mobile CPU becomes the better choice. Furthermore, the mobile CPU provides additional expansion capability to augment the existing NMI algorithm to provide actual leg control and EMG signal anomaly detection in future design iterations.

It is important to note that these results are for the phase dependent NMI algorithm and that a different algorithm will probably result in different performance and rankings for the architectures. To ensure accurate results, it is recommended that the actual target algorithm, actual architecture power utilizations during algorithm execution, and actual architecture sizes be utilized to perform this and all of the other analyses in this paper. To provide an example of how to perform these analyses, we have only taken into account the computational engine and utilized the manufacturers' maximum advertised power consumption.

2.7 Volume Constrained Analysis

This analysis will compare the computational performance of the architectures relative to the surface area that each would utilize on a circuit board assembly. We recommend that this analysis be performed for systems designs that are constrained to

operate within a small volume. Some applications that can be volume constrained are networked surveillance cameras, digital photo frames and home automation devices.

For a computational performance measure, we will utilize the number of floating point operations per second. Of interest is the computational performance of each of the architectures relative to its surface area consumption, therefore (2.3) can be used to provide a measure of the relative performance of each of the architectures for our NMI algorithm. Similarly to the power constrained analysis, the NMI algorithm utilized the same number of floating point operations per prediction; therefore, to maximize the performance of any given architecture, we must minimize the product of the prediction time and architecture surface area. Conversely, the architecture whose product of the prediction time and surface area is the largest will be the worst performing architecture for this analysis. For this analysis, the Texas Instruments TMS320C6713 DSP architecture exhibited the worst performance with a prediction time of 11.35ms and a package dimension of 27mm by 27mm. To provide a comparison between all the architectures we will take a ratio of each architecture's performance as determined by (2.3) relative to the worst performer (DSP). By comparing the architectures to the worst performer we can then provide a performance ratio utilizing (2.4).

$$\mathbf{Perf2} = \frac{\mathbf{Floating\ Point\ Operations\ For\ Each\ Prediction}}{(\mathbf{Seconds\ For\ Each\ Prediction})*(\mathbf{Area})} \tag{2.3}$$

$$\mathbf{Ratio2} = \frac{(\mathbf{DSP\ Prediction\ Time})*(\mathbf{DSP\ Area})}{(\mathbf{Architecture\ Prediction\ Time})*(\mathbf{Architecture\ Area})} \tag{2.4}$$

Table 2.2 provides the results of the power constrained analysis. As can be seen, the mobile CPU provided the highest performance, which was 53.7 times that of the DSP. Again, the mobile CPU architecture appears to be the best alternative. Furthermore, being the smallest architecture chosen for this AoA, the mobile CPU provides a viable solution for mounting the final design into the prosthesis.

2.8 Volume and Power Constrained Analysis

This analysis will compare the computational performance of the architectures using SWaP. We will examine the architectures' computational performance relative to their respective surface areas and power consumptions. We recommend that this analysis be performed for systems designs that are both volume and power constrained. Some applications that are both power and volume constrained are cell phones, tablets and neural-machine interfaces.

For a computational performance measure, we will utilize the number of floating point operations per second. Of interest is the computational performance of each architecture relative to its surface area and power consumption, therefore (2.5) can be

Table 2.2. Volume Constrained Performance Results

Computational Engine	Average Prediction Time	Surface Area	Performance Ratio
DSP TMS320C6713	11.35ms	27mm x 27mm	1X
CPU Core 2 Duo E7500	0.605ms	37.5mm x 37.5mm	9.7X
Mobile GPU GeForce 540m	0.193ms	29mm x29mm	51X
Mobile CPU Atom Z530	0.846ms	13mm x 14mm	53.7X

used to provide the relative performance of each architecture for our NMI algorithm. Similarly to the prior constrained analyses, the NMI algorithm utilized the same number of floating point operations per prediction; therefore, to maximize the performance of any given architecture, we must minimize the product of the prediction time with the architecture surface area and power consumption. Conversely, the architecture whose product of the prediction time, surface area and power consumption is the largest will be the worst performing architecture for this analysis. For this analysis, the Intel E7500 CPU architecture exhibited the worst performance with a prediction time of 0.605ms, a package dimension of 37.5mm by 37.5mm and a power consumption of 65watts. To provide a comparison between all the architectures we will take a ratio of each architecture's performance as measured by (2.5) relative to the worst performer (CPU). By comparing the architectures to the worst performer we can then provide a performance ratio utilizing (2.6).

$$\mathbf{Perf3} = \frac{\mathbf{Floating\ Point\ Operations\ For\ Each\ Prediction}}{(\mathbf{Seconds\ For\ Each\ Prediction})*(\mathbf{Area})*(\mathbf{Watts})} \tag{2.5}$$

$$\mathbf{Ratio3} = \frac{(\mathbf{CPU\ Prediction\ Time})*(\mathbf{CPU\ Area})*(\mathbf{CPU\ Watts})}{(\mathbf{Arch\ Prediction\ Time})*(\mathbf{Arch\ Area})*(\mathbf{Arch\ Watts})} \tag{2.6}$$

Table 2.3 provides the results of the SWaP analysis. As can be seen, the mobile CPU provided the highest performance, which was 163 times that of the CPU. Similarly to the prior constrained performance analyses, it is important that the results from the SWaP performance evaluations are used in conjunction with the system performance requirements prior to making a final architecture selection. Based on the

Table 2.3. SWaP Performance Results

Computational Engine	Average Prediction Time	Surface Area	Power	SWaP Ratio
CPU Core 2 Duo E7500	0.605ms	37.5mm x 37.5mm	65 watts	1X
DSP TMS320C6713	11.35ms	27mm x 27mm	1 watt	6.7X
Mobile GPU GeForce 540m	0.193ms	29mm x 29mm	35 watts	9.7X
Mobile CPU Atom Z530	0.846ms	13mm x 14mm	2.2 watts	163X

three constrained analyses and the future performance requirements of the NMI, the mobile CPU architecture appears to be the best selection.

2.9 Conclusions

This paper presented a methodology of performing constrained AoAs for the selection of computational engines for future system designs. Various analyses were utilized to evaluate power, volume and both power/volume performance constraints. Guidance was provided on when to use each analysis and how to combine the results of the analyses with performance requirements to provide the appropriate computer architecture selection for future system designs. NRE was provided for the DSP and mobile GPU architectures to aid in properly planning such an analysis. As can be seen by the three man-month effort to port and optimize the NMI for use in a mobile GPU architecture, such analysis can be time consuming and expensive. We hope that the processes and analyses presented will help other systems engineers perform their own AoAs for their system. Furthermore, we hope that the methods used for the relative performance evaluations will serve as a starting point to help shape policy in the selection of computational engines for future designs.

Our future research includes the development of multi-core variants of the phase-dependent NMI algorithm, using various programming techniques. We plan to compare the performance of multi-core processors, such as the Intel i5 and i7 architectures, to that of the mobile CPU, CPU, DSP and mobile GPU architectures. Although the size and power consumption of these architectures may exclude them from candidacy for an NMI, the additional results will provide a more complete AoA. Furthermore, the parallel capability of the multi-core processors should provide a better comparison relative to the parallel GPU architecture.

List of References

[1] C. T. Fallen, B.V.C. Bellamy, B.B. Newby, B.J. Watkins, "GPU Performance Comparison for Accelerated Radar Data Processing," Symposium on Application Accelerators in High-Performance Computing (SAAHPC), pp.84-92, 19-21 July 2011

[2] J. Siegel, J. Ributzka, L. Xiaoming, "CUDA Memory Optimizations for Large Data-Structures in Gravit Simulator," International Conference on Parallel Processing Workshops 2009 (ICPPW), pp. 174-181, 22-25 Sept. 2009

[3] A. Bustamam, K. Burrage, N.A. Hamilton, "Fast Parallel Markov Clustering in Bioinformatics Using Massively Parallel Graphics Processing Unit Computing," Ninth International Workshop on Parallel and Distributed Methods in Verification, pp. 116-125, Sept. 30 2010 – Oct. 1 2010

[4] C. Yang, Q. Wu, J. Chen, Z. Ge, " GPU Acceleration of High-Speed Collision Molecular Dynamics Simulation," Ninth International Conference on Computer and Information Technology 2009 (CIT 2009), pp. 254-259, 11-14 Oct. 2009

[5] Phys. Org, "Sun Introduces New Metric for Server Efficiency", [online]. Available: http://phys.org/news8818.html [October 2, 2012]

[6] H. Huang, F. Zhang, L.J. Hargrove, Z. Dou, D. R Rogers and K. B. Englehart, "Continuous locomotion-mode identification for prosthetic legs based on neuromuscular-mechanical fusion," IEEE Trans Biomed Eng, vol 58, pp. 2857-75, 2011.

[7] H. Huang, T.A. Kuiken, and R.D. Lipshutz, “A strategy for identifying locomotion mode using surface electromyography,” IEEE Trans Biomed Eng, vol 56, pp. 67-73, 2009.

[8] R. Hernandez, F. Zhang, X. Zhang, H. Huang, and Q. Yang, “Promise of a Low Power Mobile CPU based Embedded System in Artificial leg Control,” Conf Proc IEEE Engineering in Medicine and Biology (EMBC) 2012.

[9] J. Weston, S. Mukherjee, O. Chapelle, M. Pontil, T. Poggio, and V. Vapnik, V, “Feature selection for SVMs,” Advances in Neural Information Processing Systems 13, pp. 668 – 674, 2001.

[10] B. Hudgins, P. Parker, R.N. Scott, “A New Strategy for Multifunction Myoelectric Control,” IEEE Trans on Biomed Eng, vol. 40, no. 1, January 1993.

[11] F. Zhang, W. DiSanto, J. Ren, Z. Dou, Q. Yang, H. Huang, “A Novel CPS System for Evaluating a Neural-Machine Interface for Artificial Legs,” Proceeding of 2nd ACM/IEEE International Conference on Cyber-Physical Systems, pp. 67-76, 2011.

[12] D. Tiwari, S. Lee, J. Tuck, Y. Solihin. “Exploiting fine-grained parallelism in dynamic memory management,” IPDPS, 2010.

[13] A. Nicolau, “Loop quantization: unwinding for fine-grain parallelism exploitation,” Cornell University, 1985, Avaliable: http://ecommons.library.cornell.edu/bitstream/1813/6549/1/85-709.pdf [March 19, 2012].

[14] W. W. Hwu, P. P. Chang, “Inline function expansion for compiling C programs,” ACM SIGPLAN ’89 Conference on Programming Language Design and Implementation, Portland, Oregon, June 1989.

[15] Intel Corporation. (2010, June). “Intel® Atom™ Processor Z5xx Series Datasheet” [online]. Available: http://www.intel.com/content/www/us/en/processors/atom/atom-z540-z530-z520-z510-z500-45-nm-technology-datasheet.html [March 19, 2012]

[16] D. Marr, F. Binns, D. Hill, G. Hinton, D. Kaoufaty, J. Moller, M. Upton “Hyper-Threading technology architecture and microarchitecture,” Intel Technology Journal, vol. 6 no. 1, pp 4-15, 2002.

[17] Intel Corporation.“Ark | Intel® Core™2 Duo Processor E7500 (3M Cache, 2.93 GHz, 1066 MHz FSB)” [online]. Available: http://ark.intel.com/products/36503/Intel-Core2-Duo-Processor-E7500-3M-Cache-2_93-GHz-1066-MHz-FSB [February 18, 2013].

[18] Texas Instruments (2001, Revised 2005). “TMS320C6713 Floating-Point Digital Signal Processor Datasheet” [online]. Available: http://www.ti.com/lit/ds/sprs186l/sprs186l.pdf [February 18, 2013]

[19] Berkley Design Technology Inc., "A BDTI Analysis of Texas Instruments TMS320C67x", [online] Available: http://www.bdti.com/MyBDTI/pubs/c67_summary_report.pdf [February 18, 2013]

[20] Spectrum Digital, Inc. (2003). “TMS320C6713 DSP Starter Kit Technical Reference, Rev B”, [online]. Available: http://c6000.spectrumdigital.com/dsk6713/V1/docs/dsk6713_TechRef.pdf [February 18, 2013].

[21] NVIDIA Corporation. “Geforce Gt 540m Specifications.” [online]. Available: http://www.geforce.com/hardware/desktop-gpus/geforce-gt-540m/specifications [July 14, 2012].

[22] NVIDIA Corporation. (2012) "NVIDIA CUDA C Programming Guide", Ver. 4.2, [online]. Available: http://developer.download.nvidia.com/ compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf [September 28, 2012].

[23] G. M. Amdahl, “Validity of the single processor approach to achieving large scale computing capabilities,” AFIPS Spring Joint Computer Conference, pp. 483-485, April 1967

[24] NVIDIA Corporation. “CUDA Toolkit 4.2”. [online]. Available: http://developer.nvidia.com/cuda/cuda-downloads. [October 12, 2012].

[25] J. Nickolls, I. Buck, M. Garland. “Scalable Parallel Programming with CUDA,” ACM Queue, 6, 3, 40-53., March 2008

[26] www.techPowerUp.com. “NVIDIA GeForce GT 540M | techPowerUp GPU Database”. [online]. Available: http://www.techpowerup.com/ gpudb/702/NVIDIA_GeForce_GT_540M.html [October 12, 2012]

[27] Acer, Inc. “Aspire 4752/4752G/4752Z/4752ZG/4352/4352G Service Manual”. [online] Available: http://www.manualslib.com/download/ 232528/Acer-Aspire-4352.html [February 20, 2013]

MANUSCRIPT 3

Towards Ubiquitous Mobile-Computing-Based Artificial Leg Control

by

Robert Hernandez[1], Jason Kane[2], Fan Zhang[2],

Xiaorong Zhang[2], and He Huang[2]

is submitted to IEEE Transactions on Mobile Computing.

[1] Robert Hernandez is with Torpedo Systems Department, Naval Undersea Warfare Center, Newport, RI, 02842, Email { robert.hernandez2, john.faella }@navy.mil.

[2] Jason Kane, Fan Zhang, Xiaorong Zhang, and He Huang are with Department of Electrical, Computer, and Biomedical Engineering, University of Rhode Island, Kingston, RI, 02881, Email { jkane, fzhang, zxiaorong, huang, }@ele.uri.edu.

Abstract

This paper presents a rapid prototype approach for the development of a real-time capable neural-machine-interface (NMI) for control of artificial legs based on mobile processor technology (Intel Atom™ Z530 Processor.) By effectively exploiting the architectural features of a mobile embedded CPU, we implemented a decision-making algorithm, based on neuromuscular-mechanical fusion and gait phase-dependent support vector machines (SVM) classification to meet the demanding performance constraints. To demonstrate the feasibility of a real-time mobile computing based NMI, real-time experiments were performed on an able bodied subject with window increments of 50ms. The experiments showed that the mobile computing based NMI provided fast and accurate classifications of four major human locomotion tasks (level-ground walking, stair ascent, stair descent, and standing) and a 46X speedup over an equivalent MATLAB implementation. The testing yielded accuracies of 96.31% with low power consumption. An offline analysis showed the accuracy could be increased to 98.87% with minor modifications to the application.

3.1 Introduction

In 2005, there were approximately 1.6 million people in the United States with some kind of limb loss [1]. By the year 2050, the number is expected to increase to 3.6 million people [1]. Furthermore, in 2005, lower limb loss accounted for almost two-thirds (65.5%) of the 1.6 million [1]. People with lower-limb amputations typically favor their intact limb and therefore provide additional stress upon their intact limb during everyday activities [2]. It has been speculated that the additional stress placed upon their intact limb will lead to degenerative diseases [2]. These statistics clearly present the increasing need for technology that restores as much functionality to the large and increasing population of lower limb amputees.

The recent development of powered artificial legs, such as the Power Knee [3] and the Vanderbilt University design [4], provide positive mechanical energy that helps restore the user's locomotion modes [5]. These devices detect the user's intended locomotion mode though the use of echo control or solely though intrinsic mechanical feedback. In particular, the Power Knee [3] utilizes echo control [4] and requires instrumentation of the sound leg in order to detect what locomotion mode the user is currently performing. The system described in [4] utilizes, solely, intrinsic mechanical feedback [6]. In contrast, we have developed a Neural Machine Interface based on neuromuscular-mechanical fusion [7] and phase-dependent pattern recognition (PR) strategy [8]. Our strategy does not require instrumentation of the sound leg and has been shown to provide higher accuracy than the classifiers utilizing only electromyographic (EMG) data or only mechanical data [9]. Our PR strategy can be implemented utilizing either Support Vector Machines (SVM) or Linear

Discriminant Analysis (LDA) classifiers. The selection of a Support Vector Machines (SVM) classifier provided improved prediction accuracy performance of our PR strategy when compared to a Linear Discriminant Analysis (LDA) classifier [7]; therefore for this study we will utilize an SVM-based classifier.

In order to make our PR strategy a feasible reality we developed a Cyber Physical System (CPS), designed to test our Neural-Machine Interface (NMI). This CPS is a unique and complex system consisting of biomedical engineering components, a mechanical prosthesis, as well as computer software and hardware. Our objective here is to integrate various components in such a complex system in an optimal way using a system engineering approach. The important parameters that we aim to optimize include mainly 1) real-time performance to provide fast control of prosthesis; 2) high accuracy of locomotion prediction; 3) low power consumption; and 4) small size wearable by leg amputees.

With these objectives in mind, we investigated commercial off-the-shelf (COTS) computing devices and chose one ubiquitous mobile computing system, the Intel AtomTM Z530. It is low power (2.2 watts [10]), low cost, and a portable mobile computer that meets our NMI performance requirements. Our preliminary study [11] showed that a mobile processor based NMI had great promise in control of artificial legs [11]. The primary objective of this paper is to determine the viability of mobile technology as a possible architectural solution for use in our 50ms window increment NMI. We chose to utilize 50ms window increments in this study to provide a comparison with our existing MATLAB implementations. Additionally we wish to determine if the Intel Atom based design will allow for further expansion of our NMI

algorithm to perform electromyographic (EMG) anomaly detection and perform the prosthesis leg control by sending control signals based on our PR strategy at 10ms intervals, it is desirable to quantitatively evaluate the mobile technology's reserve capability while executing our 50ms window increment NMI.

Existing solutions for prosthesis control have been implemented on MATLAB that cannot satisfy real-time requirements running on the mobile computing device. We have developed an entire software implementation of our SVM-based PR strategy in C to run on the mobile computer. It turns out that porting the software to the mobile computer present several challenges to meet our goals. The first challenge is the time constraint of the NMI to deliver correct control decision in real time. Straight forward implementation is far from satisfactory. We therefore proposed several innovative techniques to exploit the inherent architectural features, which are described in detailed in Section 2. Another challenge is low power consumption. We proposed implementation techniques that can lower CPU requirements so that power consumption is kept minimal.

To meet our research objectives, we designed and developed a real time software interface to a data acquisition system (DAQ) providing the capability to acquire real-time EMG, mechanical force and moment data from human subjects, with no data loss or lag. This newly developed NMI was combined with a Measurement Computing USB-1616HS-BNC DAQ [12] to facilitate the collection of the real-time EMG and 6 degrees-of-freedom (DOF) mechanical data. This final NMI design was utilized to execute and test the real-time performance of our phase dependent SVM based PR algorithm utilizing 50ms window increments on an able bodied human subject.

This paper makes the following contributions:

- Design and implementation of a real-time capable NMI utilizing 50ms window increments for artificial leg control based on a mobile processor;
- Design and implementation of a highly optimized program for a phase-dependent NMI with SVM classifiers tailored specifically to the mobile processor utilizing 50ms window increments;
- A comparison between our new C based NMI embedded application and our equivalent MATLAB based NMI that shows the embedded C application provides a 46X speedup;
- A real time experiment that evaluates the potential use of mobile processors for a 50ms window increment embedded implementation for neural control of powered lower limb prosthesis;
- An analysis that shows the future algorithm expansion capability of this mobile based NMI implementation.

This paper is organized as follows. Next section presents an expanded description of our previously published offline system design [11]. Sections 3 and 4 present our previously published pattern recognition algorithm and offline performance evaluation. Sections 5, 6, 7, and 8 present our newly designed and developed 50ms window increment real-time system design, software implementation, experimental protocol and performance evaluation. Section 9 presents recommended updates to our new real-time algorithm and updated performance expectation. We conclude our paper in Section 10.

3.2 Offline Study System Design

3.2.1 Hardware Architecture

To provide viable capability of prosthesis control, the NMI must be small, dissipate low power, and be fast enough to execute the classification algorithm in real-time. One possible candidate chosen to meet these requirements is the Intel AtomTM Processor Z530 (512K cache, 1.6 GHz) single core CPU [10]. The AxiomTek eBOX530-820-FL [13] fanless embedded hardware was chosen as the COTS prototype architecture to test the viability of the Intel AtomTM Processor. The Intel AtomTM Processor Z530 provided the highest performance and lowest power dissipation of available hyper-threading capable mobile CPUs, which is ideal for thermally constrained and fanless embedded applications [10, 14]. The Hyper-Threading technology provides the capability for the operating system and the NMI application to execute simultaneously on their own Hyper-Threads providing similar capability to that of executing on two physical processors, when only a single processor is utilized [15]. This helps to minimize the impacts of the OS execution on the real time embedded NMI application.

3.2.2 Software Architecture

We have developed the entire SVM-based NMI application in C because of its superior performance for real-time embedded applications [16-19]. To enhance the system performance, several programming techniques were used in the design and implementation of the application. For example, dynamic memory management is one of the most expensive operations in C applications [20]. In fact, it has been shown that heap intensive C applications, on the average spend 30% of the execution time in

dynamic memory management [20]. To avoid execution time spent on dynamic memory management, the various data structures within the software were defined utilizing arrays with pre-defined maximum sizes. To increase the reliability of the application and help avoid any stack overflows, the data structures were defined as "static." Static variables are placed in an application's data segment, not in the application's stack [21], hence avoiding stack overflows, push/pop penalties and increases the applications reliability. Other performance enhancements implemented were loop unwinding [22] and inline function expansion [23]. Loop unwinding is an efficient means to increase the utilization of pipelines and helps eliminate loop overhead [22]. For example, if the number of times a loop will execute is known prior to the body of the loop and the control code can be duplicated, thereby eliminating the loop overhead [22] and mitigating any pipeline stalls due to branch hazards [24]. The feature extraction code is one computationally intensive area where loop unwinding was utilized. The feature extraction code was highly repetitive and the number of raw data channels and features per channel were known ahead of time, which made it an excellent candidate for loop unwinding. A simple example of loop unwinding is shown in Fig. 3.1, whereby all the j variable comparisons and the need for branch prediction to determine when the j loop has completed are eliminated via loop unwinding. Upon further examination of Fig. 3.1, it can be seen that the i loop can also be unwound. Since variable i iterates a total of 150 times (window length), the resultant code would become unmanageable. Therefore, an engineering tradeoff between performance and software maintainability led to the decision to not unwind the i loop code. For our PR algorithm, the loop unwound code's execution time was

Original Code:

```
for (j = 0; j < 7; j++)
{
    for (i = 0;i < WindowLength;i++)
    {
        CH_Mean[j] += *(channels[j] + i + start_index);
    }
    CH_Mean[j] /= WindowLength;
}
```

Unwound Code:

```
for (i = 0;i < WindowLength;i++)
{
    CH_Mean[0] += *(channels[0] + i + start_index);
    CH_Mean[1] += *(channels[1] + i + start_index);
    CH_Mean[2] += *(channels[2] + i + start_index);
    CH_Mean[3] += *(channels[3] + i + start_index);
    CH_Mean[4] += *(channels[4] + i + start_index);
    CH_Mean[5] += *(channels[5] + i + start_index);
    CH_Mean[6] += *(channels[6] + i + start_index);
}
CH_Mean[0] /= WindowLength;
CH_Mean[1] /= WindowLength;
CH_Mean[2] /= WindowLength;
CH_Mean[3] /= WindowLength;
CH_Mean[4] /= WindowLength;
CH_Mean[5] /= WindowLength;
CH_Mean[6] /= WindowLength;
```

Figure 3.1. Simple example of loop unwinding to calculate channel means

approximately 10% faster than the original code; these results were with the compiler speed optimization enabled for both the original and the unwound code.

Inline function expansion replaces a function call with the body of the function [23]. This reduces the overhead associated with a function call during program execution [23]. Because the keyword inline only serves as a hint to compilers and not all compilers support the inline keyword [23], to further reduce overhead the total number of function calls were kept to a minimum.

The Neuromuscular-Mechanical fusion PR algorithm, utilizes SVM for its classification. Our prior studies based on the same Neuromuscular-Mechanical fusion PR recognition utilized the MATLAB release version of LIBSVM [25], which provided high accuracy. Analysis of the LIBSVM source showed that it could be possible to modify the libraries for real-time use. Therefore, the open source library LIBSVM was used and specifically tailored to our embedded NMI application for real-time SVM classification. This was beneficial since, in addition to its high accuracy, it also allowed LIBSVM to serve as a baseline for accuracy determination of the embedded application.

3.3 Pattern Recognition Algorithm

The previously developed NMI identifies the user's locomotion mode based on electromyographic (EMG) signals recorded from the residual thigh muscles and mechanical forces/moments signals recorded from prosthetic pylon. These EMG and mechanical data are segmented by the sliding analysis windows. Features are extracted from the raw EMG and mechanical data in each analysis window and fused into one feature vector. This feature vector is sent to a phase-dependent pattern classifier for determination of user intent. The phase-dependent pattern classifier consists of multiple sub-classifiers for individual defined gait phases and a gait phase detector that identifies current gait phase and switches the corresponding sub-classifier on. Detailed description of this previously designed NMI can be found in [7] and [8].

3.3.1 Feature Extraction

In this study, four time-domain (TD) features (the mean absolute value, the number of zero crossings, the waveform length, and the number of slope sign changes)

were extracted from EMG signals in each analysis window. For mechanical measurements, the mean, minimum, and maximum values in each analysis window were extracted as the features. More detailed information can be found in [7]. The length of sliding analysis window and window increment were 150ms and 50ms, respectively.

The features and increments were chosen to match our previous MATLAB implementations [26], thereby providing a baseline for an accuracy comparison with the newly designed embedded application.

3.3.2 Phase Dependent Pattern Recognition

To accurately determine user intent, an SVM based classification architecture utilizing a Radial Basis Function (RBF) kernel and an SVM gamma parameter of 0.015 was employed [7, 8]. The phase-dependent classifier is composed of four sub-classifiers corresponding to one of the following four gait phases: initial double limb stance (phase 1), single limb stance (phase 2), terminal double limb stance (phase 3), and swing (phase 4) [26]. Throughout this paper, inclusive of the figures, we utilize the following gait phase definitions: 1 - Initial Double Limb Stance, 2 - Single Limb Stance, 3 - Terminal Double Limb Stance and 4 - Swing. The gait phase detector uses the real-time vertical Ground Reaction Force (GRF) to determine the gait phases. In order to build the SVM sub-classifier models, a training procedure is conducted on all the acquired training data sets. During training phase, the output of the phase detector is used to label the training data with its corresponding gait phase. Each sub-classifier is trained only with the data pertinent for its gait phase. During the real-time testing phase, the gait phase detector determines which sub-classifier is responsible for the

determination of user intent. The gait phase detector's determination is used to select the appropriate sub-classifier to act upon the feature vector composed of fused EMG and mechanical data. The algorithmic data flow of the phase-dependent pattern recognition is shown in Fig. 1.2.

3.3.3 Software Implementation

To implement the Neuromuscular-Mechanical Fusion PR, three applications were developed. The first application accepts offline raw training data, performs the EMG and mechanical feature extraction, fuses and then normalizes the features into vectors. The feature vectors are then separated into their corresponding gait phases and provided to the training application. The first application is also responsible for generating the normalization parameters required by the PR to normalize the testing data, when determining user intent. The second application accepts the four sets of training vectors and generates four SVM models, one model for each gait phase. The third application accepts raw offline testing data, the four gait phase SVM models, and the normalization parameters. The application extracts EMG and mechanical features from the raw testing data. The features are then fused and normalized, with the provided normalization parameters, into a vector. Finally, the application determines the current gait phase, and forwards the test vector to the respective phase based classifier for determination of user intent. The offline analysis software implementation data flow is shown in Fig. 1.3.

3.4 Offline Performance Evaluation

All experiments performed in this study were conducted with the approval of the Institutional Review Board (IRB) at the University of Rhode Island and with the

informed consent of the subject. The evaluation was performed offline on the data collected from a male subject with a transfemoral amputation. The collected data included the EMG signals from the subject's residual thigh muscles and mechanical forces/moments measured by a 6 degree-of-freedom load cell mounted on the prosthetic pylon. The monitored residual muscles included the rectus femoris (RF), vastus lateralis (VL), vastus medialis (VM), biceps femoris long head (BFL), semitendinosus (SEM), biceps femoris short head (BFS), and adductor magnus (ADM). The recognition accuracy of NMI by using the designed embedded system was compared with the results of existing PC implementations on MATLAB. In addition, the timing and processor loading of the application's execution on the embedded hardware were evaluated. A power consumption comparison between similar proposed NMI embedded systems and this embedded system was provided.

3.4.1 Recognition Accuracy of NMI

The offline data was composed of seven different classes: level-ground walking (W), ramp ascent, ramp descent, stair ascent (SA), stair descent (SD), sitting, and standing (ST). The comparison of recognition accuracies of the NMI by using the designed embedded system and existing PC implementations on MATLAB are provided in Table 1.1. This study utilized a slightly different value for the gamma parameter required by the SVM classifiers. The different gamma value was shown to provide a slightly higher accuracy during testing. This is noticeable in the comparison results, whereby the embedded application slightly outperformed the MATLAB model in PR accuracies.

Both the MATLAB results and the embedded application had lower Phase 4 (swing) accuracies. Two explanations for this result are provided in [8]. The first is that there is little force/moment data present during the swing phase from the prosthetic pylon [8]. The second explanation is related to the swing phase being longer than any of the other three phases, leading to larger variations in the EMG features [8].

3.4.2 Execution Timing and Processor Loading on the Embedded Hardware

This previously designed NMI algorithm was executed on the Intel Atom™ based embedded hardware and the performance results were evaluated. A total of 3555 predictions were produced by the Intel Atom™ based embedded hardware. For the purpose of this evaluation, the prediction time will be defined as the total time to execute feature extraction, normalization, gait phase detection and classification for a single analysis window. The mean prediction time was 0.8455 milliseconds with a standard deviation of 0.1044 milliseconds. The worst case prediction executed in 2.1265 milliseconds. These results clearly show that the embedded system is capable of real-time implementation at 50ms and 20ms window increments. If the embedded system is combined with a highly responsive Data Acquisition (DAQ) system to provide the EMG and mechanical data, even a window increment of 10ms may be feasible. At the 10ms window increment, the interface to the DAQ and the DAQ system drivers will become of the utmost importance.

Because there is additional loading on the CPU to execute the data logging for post analysis, the CPU loading provided by the operating system may be inaccurate. Therefore the mean and maximum value of the CPU loading was calculated by (3.1) to be 1.691% and 4.253% respectively.

$$\textbf{CPU Loading} = \frac{\textbf{Prediction Time}}{\textbf{Window Increment (50ms)}} * \textbf{100} \quad (3.1)$$

3.4.3 Power Consumption Comparison

Previous studies have utilized Field Programmable Gate Arrays (FPGA) and PCs for similar NMI applications [27]. The reported power consumption for the FPGA was 3.499 watts and the AMD Turion 64x2 CPU within [27] can utilize up to 35 watts [27]. The Intel Atom™ Z530 Processor utilized in this embedded system design dissipates 2.2 watts maximum [10]. The Intel Atom™ CPU's power dissipation is less than one-fifteenth that of the AMD CPU and less than two thirds that of the FPGA.

3.5 Real-Time Capable System Design

Based on the offline performance and the results of our Analysis of Alternatives (AoA) [28], it was decided to continue using the AxiomTek eBOX530-820-FL fanless embedded hardware with the Intel Atom™ Processor Z530 (512K cache, 1.6 GHz) as our COTS mobile computing system. During the source selection of a DAQ to combine with the AxiomTek embedded hardware, it was clear that the vast majority of COTS DAQ devices with the capability to meet our design requirements (16 analog input channels and simultaneous sampling or a similar capability) only provided drivers for the Windows and Linux operating systems. The NMI design needs to meet real-time constraints and therefore the use of a Real-Time Operating System (RTOS) is preferable. An RTOS performs its functions, including external events in a specified amount of time [29]. Windows and Linux are general purpose operating systems (OSs) and do not meet the criteria of an RTOS. Therefore, as a compromise, it was decided to utilize a general purpose operating system with the understanding that RTOS

options were available for both Windows and Linux implementations, such as Windows Compact Embedded (WinCE) [30] and Real-Time Linux (RT Linux) [31]. Furthermore, it would be expected that if real-time constraints can be met with a general purpose OS, then porting the design to an RTOS would provide better system response and make the design more deterministic. The decision to go with the Windows OS vs. Linux was based on the experience and familiarity of the research team with the Microsoft Visual Studio product. This familiarity would facilitate the rapid design, implementation and debugging of the prototype COTS solution.

For our COTS prototype, Measurement Computing's USB-1616HS-BNC DAQ was chosen to interface with the AxiomTek eBOX530-820-FL fanless embedded hardware to provide the real-time EMG and loadcell data necessary to make our neuromuscular-mechanical fusion SVM NMI a feasible reality. The Measurement Computing device met all our performance requirement, provided a C-library interface that was capable of interfacing with our prior embedded software design, and was easily interfaced to the AxiomTek embedded hardware via a universal serial bus (USB) port.

3.6 Real-Time Capable Software Implementation

All of the initial software architectural and implementation decisions made in our design, such as the use of the C programming language, loop unrolling and inline function expansion were utilized within the real-time implementation. In addition a few other techniques were incorporated to augment and provide further performance enhancement.

3.6.1 Software Architecture

The use of a general purpose OS in this prototype design iteration raised concerns with the embedded software's capability to meet real time constraints. Therefore, to further reduce the impact of the OS on the embedded application, the priorities of the application and thread were increased to a real time critical status. In a Microsoft Windows OS, this is accomplished by setting the priority class to REALTIME_PRIORITY_CLASS and the thread priority to THREAD_PRIORITY_TIME_CRITICAL [32].

The real-time software implementation required that all raw data, phase data, and classification data be logged to allow for performance evaluations. To minimize the impacts of the real-time data logging on the application, a statically allocated and statically defined Random Access Memory (RAM) buffer was implemented that stored all the raw EMG, mechanical, classification and application performance data. The RAM buffer eliminated the need to write to the hard drive during time critical operations. Furthermore, it took advantage of the RAM's superior speed for storage. The real-time data logging for each classification was performed after all time-critical functions were completed (i.e., at the end of each classification). Lastly, The RAM buffer's contents were written to the hard drive for post analysis after the experiment was completed, by which point no further time critical functions were being executed.

Re-implementing the our software optimizations and the newly incorporated additional enhancements, resulted in an embedded application specifically designed to minimize pipeline stalls, minimize OS impacts, minimize cost of memory allocation, minimize the impacts of real-time data logging and take advantage of the Intel

Atom™ Z530 Processor hardware architecture. These enhancements provided the basis for the performance introduced by this embedded application.

3.6.2 Real-Time Software Implementation

To implement the real-time Phase-Dependent PR algorithm, four applications were required. Where previously the offline study's data was read in via a file, the real-time study requires a new application to be developed to interface with the DAQ and capture real-time training data. The feature extraction & normalization application, as well as the SVM training application remained unchanged. Finally, the Neuromuscular-Mechanical Fusion PR application had to be modified to acquire real time data testing from the DAQ. The training data capture application acquires data for all of the various human locomotion tasks, segregates the data into each locomotion class, and allows for multiple trials of each locomotion task. The real-time PR application is used during the real-time testing phase. The real-time PR application extracts EMG and mechanical features from the raw testing data acquired in real-time from the DAQ. Similarly to the offline method, the features are then fused and normalized with the provided normalization parameters and formed into a vector. Finally, the application determines the current gait phase, and forwards the test vector to the respective phase based classifier for determination of user intent. The software implementation data flow is shown in Fig. 3.2.

3.7 Real-Time Experimental Protocol

A real-time performance evaluation utilizing a 50ms window increment and an offline performance evaluation utilizing a 50ms window increment were performed as part of the real-time study. The evaluations were performed on the data collected from

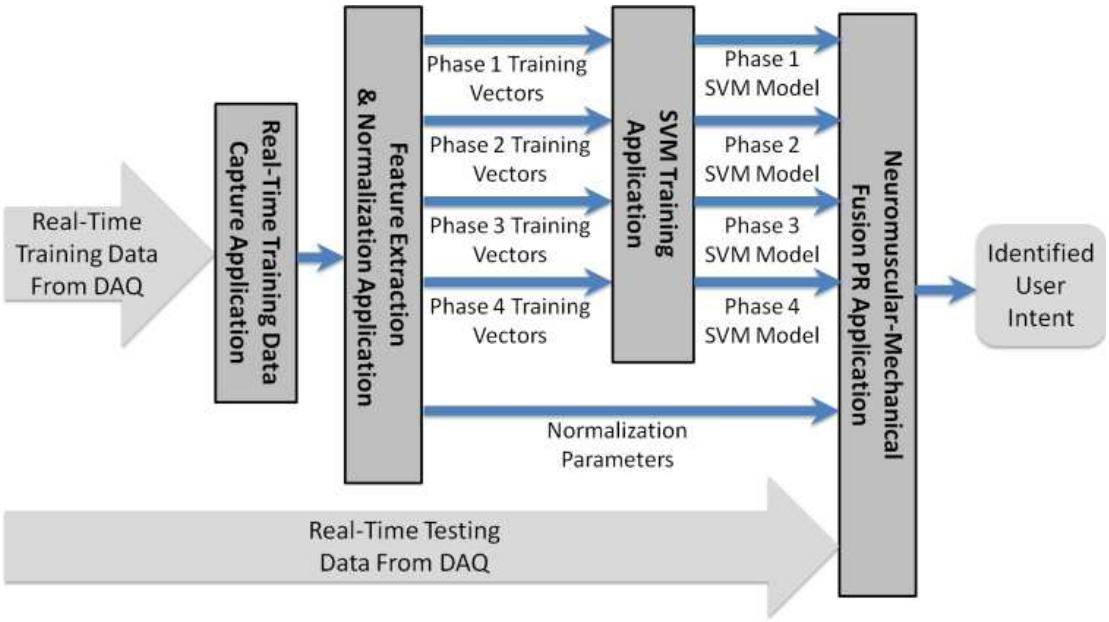

Figure 3.2. Real-time software implementation data flow

a male able bodied subject. The collected data included the EMG signals from the subject's thigh muscles and mechanical forces/moments measured by a 6 degree-of-freedom load cell mounted on the prosthetic pylon. The monitored muscles included the sartorius (SAR), rectus femoris (RF), vastus medialis (VM), adductor magnus (ADM), biceps femoris short head (BFS), biceps femoris long head (BFL), and semitendinosus (SEM).

The EMG and mechanical forces/moments were sampled at 1 KHz by the Measurement Computing USB-1616HS-BNC DAQ device. The user intent decisions provided by the embedded hardware were routed via an analog output interface on the DAQ device. The real-time experiments provided real-time gait-phase and user intent decisions to the console screen as a visual cue during the training and testing processes. The 50ms window increment experiments utilized a window increment of 50ms and a window length of 150ms.

For all experiments performed in this study, the prediction time will be defined as the total time to execute feature extraction, normalization, gait phase detection, majority vote (if performed) and classification for a single analysis window.

3.8 Real-Time Performance Evaluation

For this experiment, four tasks (level-ground walking (W), stair ascent (SA), stair descent (SD), and standing (ST)) were studied and captured for offline analysis. To ensure the subject's safety, the subject was allowed to use hand rails when necessary. To train the gait-phase classifier, the subject was instructed to perform each task for approximately 10 seconds. Two trials of standing data, three trials of walking data, three trials of stair descent and three trials of stair ascent data were accumulated to train the classifier. For the real-time performance evaluation, the subject was instructed to stand and then transition to one of the other tasks (ST→W, ST→SD and ST→SA). Seven trials of each mode transition were conducted, for a total of 21 trials. To assess the real-time performance of the NMI, the timing and processor loading of the application's execution on the embedded hardware are provided and the raw recognition accuracy of the NMI will be evaluated via the following criteria:

Classification Accuracy in the Static States: For all experiments in this paper, the static state is defined as the state where the subject has completed a transition and is continuously performing the same task (W, SA, SD or ST). The classification accuracy in the static state is the total number of correct classifications observed over the total number of classifications during the static state.

The overall raw classification accuracy of the NMI in the static states for all 21 trials and all tasks (W, SA, SD and ST), when executed on the Intel Atom^{TM} based

embedded hardware was 96.31%. A total of 5937 static state predictions were produced by the Intel Atom™ based embedded hardware during the 21 trials. The mean prediction time for all of the predictions performed during the 21 trials was 0.7683ms with a standard deviation of 0.0971ms. The worst case prediction executed in 2.0192ms.

Due to the fact that there is additional loading on the CPU to execute the data logging for post analysis, the CPU loading provided by the operating system may be inaccurate; therefore the mean and maximum values of CPU loading were calculated using Equation (3.1), which were 1.54% and 4.04% respectively. These results show that the majority of the time, the embedded software design was awaiting new EMG and Loadcell data from the DAQ, as shown in Fig. 3.3. During this time the processor is idle and can be utilized to execute other additional algorithms to augment our NMI's capability.

Although 96.31% accuracy is very good, it fell short of the average 97% accuracy that was achieved by the MATLAB model in the offline analysis shown in Table 1.1. Furthermore, based on the offline analysis, this implementation was expected to perform approximately 1% higher than the MATLAB model due to the use of a different SVM gamma value. Upon further review of [26], it became obvious that this 50ms window increment embedded software design did not incorporate a real-time majority vote method. Upon examination of the raw data, it was apparent that a 5-point majority vote method could have a substantial effect on the overall system accuracy. For example, in Fig. 3.4 we see a real-time stair ascent trial with 6 misclassifications. We manually post processed the stair ascent data, implementing the

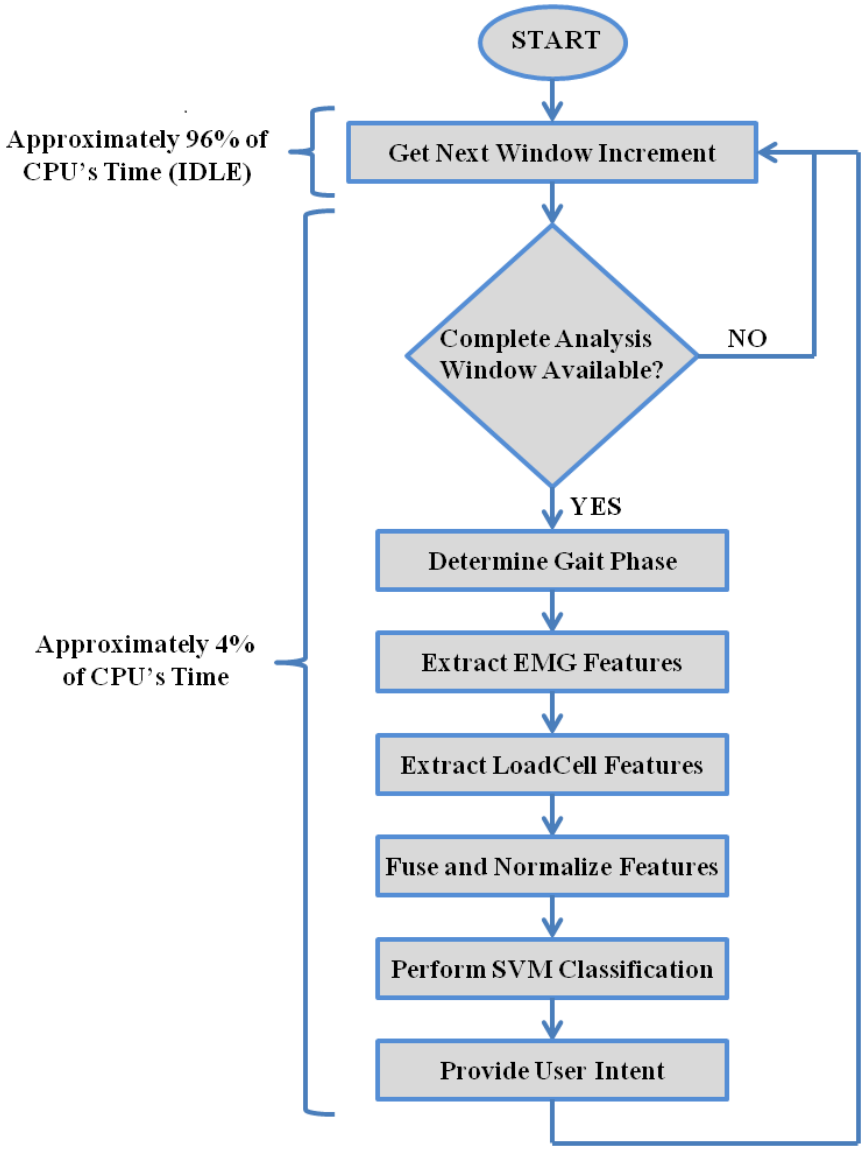

Figure 3.3. Simplified real-time software flowchart and CPU utilization

5-point majority vote, which led to the removal of all misclassifications as shown in Fig. 3.5. The implementation of a majority vote increased the accuracy from 97.9% to 100% for this trial. In order to determine if this was the cause for the discrepancy in overall accuracy, it was decided to perform an offline analysis of this algorithm with a majority vote implementation.

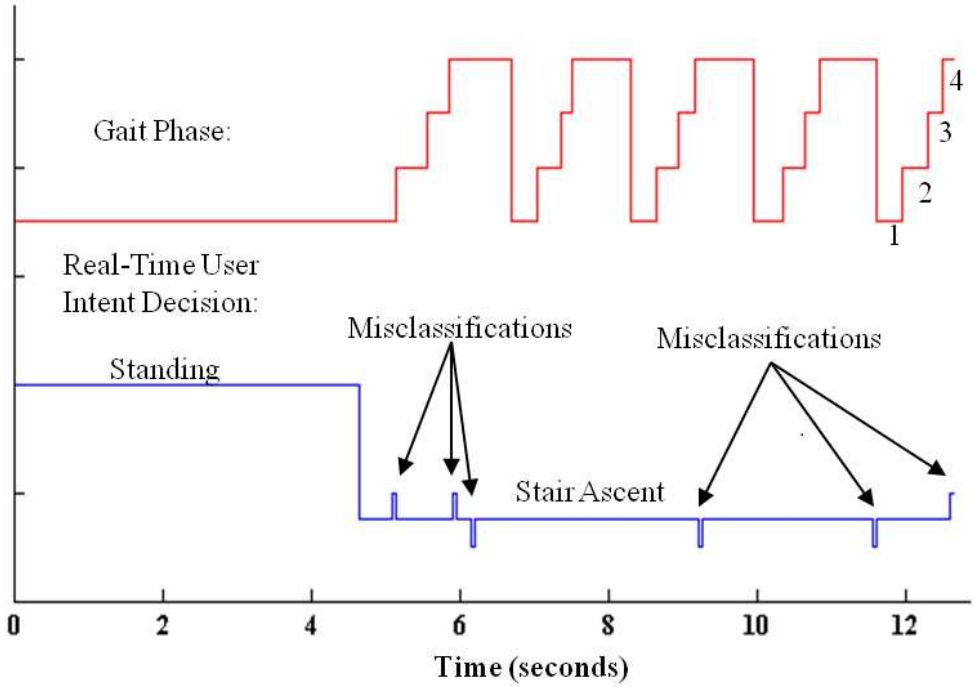

Figure 3.4. Real-time stair ascent trial showing misclassification prior to majority vote implementation

3.9 Modified Real-Time Algorithm Evaluation

The 50ms window increment offline evaluation utilized the exact data acquired during the real-time experiment. This allows for an accurate comparison between the original software design and this proposed design.

To perform this evaluation, the initial software was modified to utilize the raw DAQ data logged during the real-time testing. The algorithm was further modified to provide a five-point majority vote algorithm as in [26]. For this experiment, the same four tasks (W, SA, SD, and ST) were examined. Since the intent of this study is to determine the mobile CPU's capability to execute our PR algorithm, initially it was determined that examining the Classification Accuracy in the Static States should suffice. However, since slight modification to the software would enable mode transition performance evaluations that initiate from a standing position and all of the

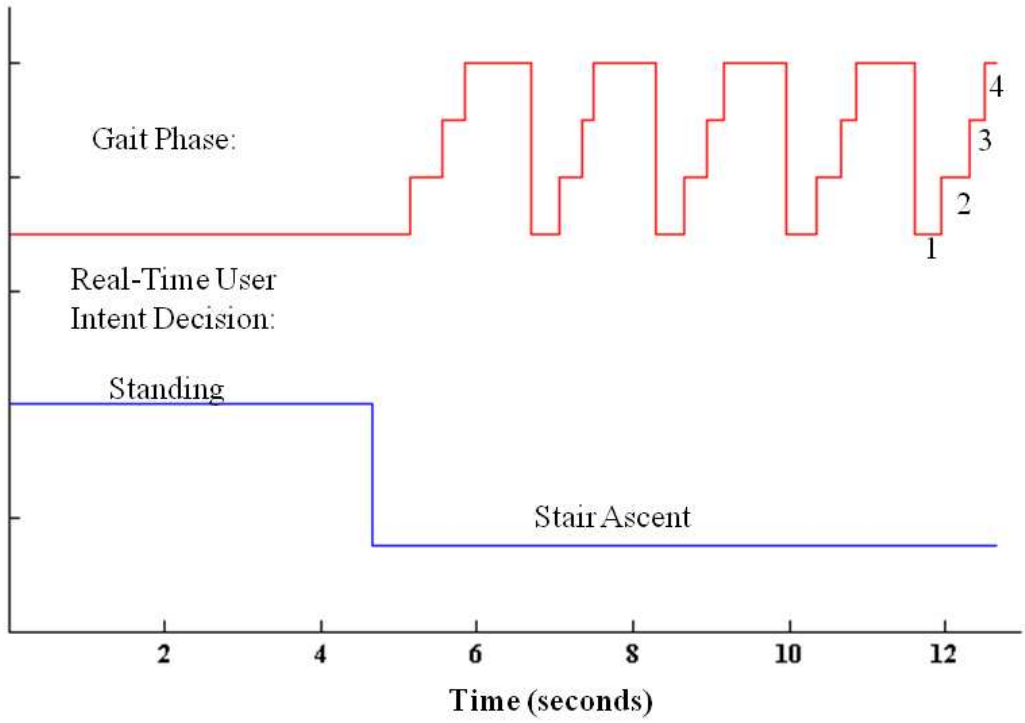

Figure 3.5. Stair ascent trial manually post processed with a 5-point majority vote showing no misclassifications

raw data was recorded during the real-time trials, we were also able to examine the performance during the three mode transitions (ST→W, ST→SD and ST→SA), therefore the analysis was performed and the results have been included within this paper. Additionally, included in the offline evaluation is a speedup assessment of the C based embedded application to the MATLAB based application. The performance of the NMI will be evaluated using the following criteria:

Classification Accuracy in the Static States: As previously defined in the real-time 50ms experiment.

The Number of Missed Mode Transitions: For this experiment, the mode transition period starts from the beginning of gait phase 2 (single limb stance) and

terminates at the beginning of gait phase 4 (swing). A mode transition is declared to have been missed, if no correct transition decision is made during this defined period.

Mode Transition Prediction Time: The mode transition prediction time in this experiment is defined as the amount of time prior to the critical timing, during which the classifier user intent decision has stabilized and is no longer changing, such that safe switching of the prosthesis device is made possible. For this experiment, the critical timing is defined as the termination of the mode transition (i.e. - just prior to the start of the swing gait phase).

3.9.1 Speed Up Provided by the C Embedded Application

A self-contained version of the PR embedded application was built with raw test data resident within the application itself. Timing analysis software was added to verify the performance of the embedded software design and implementation. To provide an accurate comparison between the MATLAB based NMI and our C based embedded application, our application was executed on the MATLAB system for a determination of average prediction time. The MATLAB system is composed of Core 2 Duo E7500 CPU clocked at 2.93 GHz with 3GB of RAM and executes the Windows XP operating system. A total of 1002 classifications were performed by the PR embedded application on the MATLAB system and completed in 472.53ms. This results in an average of 472 microseconds per classification. The average classification time of the MATLAB model executed on the same system was 21.9ms. Based on this experiment, the C based embedded application provides a 46X speedup over the MATLAB model.

Although this is obviously not an “apples to apples” comparison (i.e. - MATLAB vs. C), this does provide critical information that is useful to systems engineers; it allows them to understand what speedup can be achieved and/or expected by simply porting a MATLAB algorithm to an optimized C-based application. We would like to have provided a comparison of our PR algorithm on other embedded architectures, but this is our first embedded implementation, therefore no other comparison is available

3.9.2 Recognition Accuracy of NMI

The overall classification accuracy of the NMI in the static states for all 21 trials and all tasks (W, SA, SD and ST) was 98.87%. No missed mode transitions were observed during the defined mode transition period. The mean mode transition prediction time for ST→SA was 871.4ms with a standard deviation of 197.6ms. The mean mode transition prediction time for ST→W was 528.6ms with a standard deviation of 107.5ms. The mean mode transition prediction time for ST→SD was 314.3ms with a standard deviation of 94.5ms. The mode transition performance implies that user intent classification during transitions can be accurately determined, on the average, 314.3ms prior to the critical timing and be used for safe switching and control of the prosthesis. Representative trials, depicting the user intent classifications prior and during the ST→SD, ST→SA and ST→W mode transitions are provided in Figures 3.6, 3.7, and 3.8, respectively. As can be seen in Figs. 3.6 thru 3.8, there were a few misclassifications during the ST→W and ST→SD transitions, but it can be seen that the transitions were correctly predicted prior to the critical timing and the static state accuracy was 100% during these three trials.

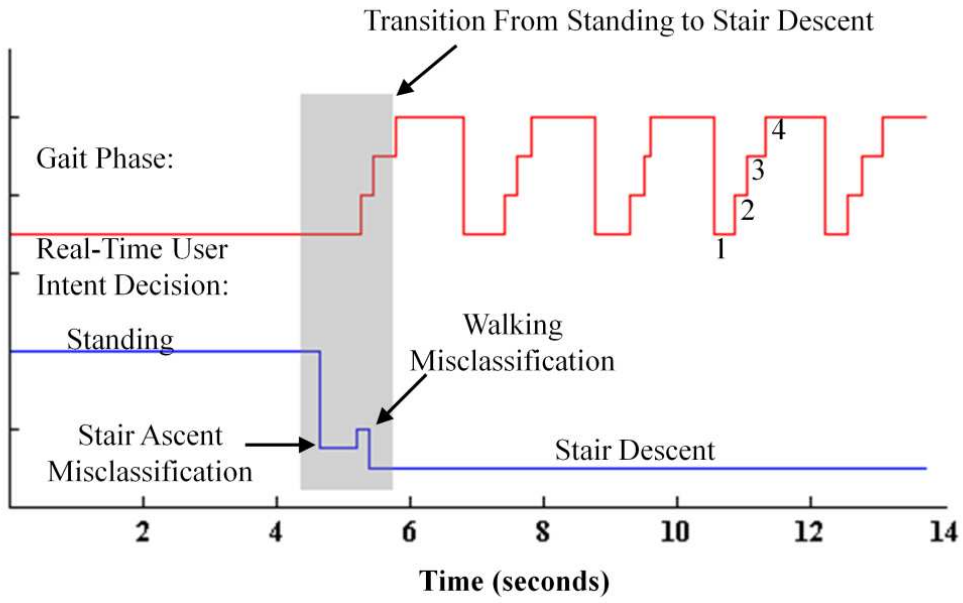

Figure 3.6. Offline performance of a standing to stair descent trial

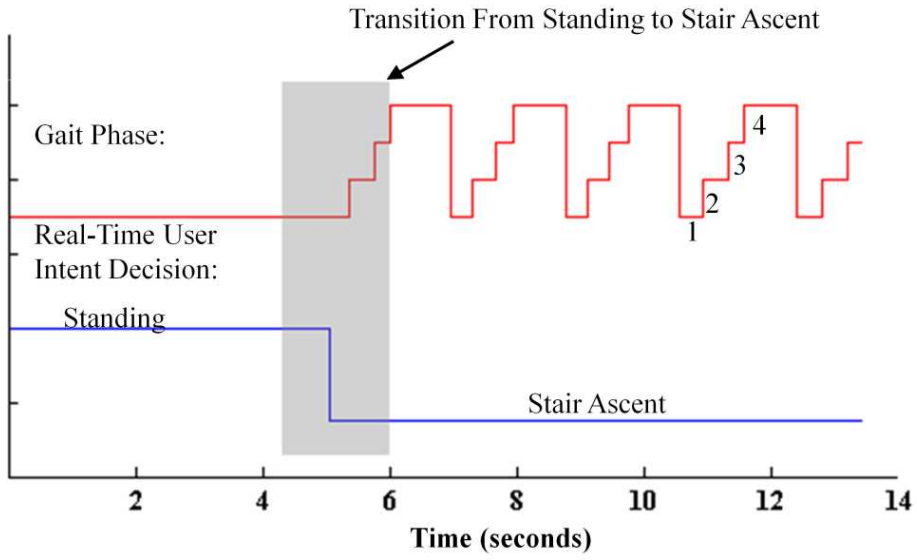

Figure 3.7. Offline performance of a standing to stair ascent trial

This revision to the algorithm provided an additional 2.5% accuracy in static states, while still meeting all of its other performance requirements. This clearly

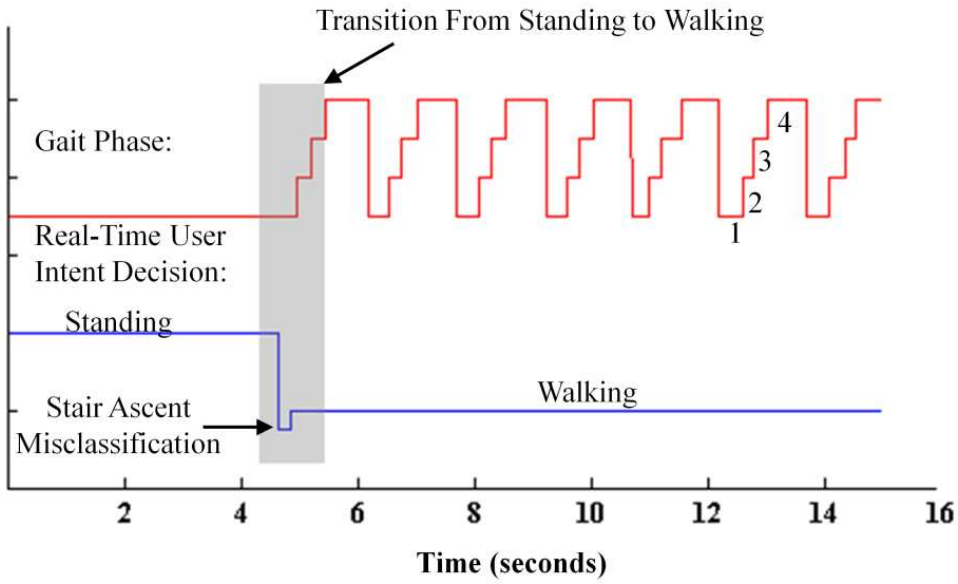

Figure 3.8. Offline performance of a standing to walking trial

showed that the majority vote method is a critical component of the algorithm and must be included in future implementations and/or expansion of the algorithm.

3.10 Conclusions

This paper presented the design and implementation of a mobile CPU based neural machine interface for artificial legs. The designed NMI prototype was tested on an able-bodied subject for classifying multiple movement tasks (level-ground walking, stair ascent, stair descent and standing) in real-time. The 50ms window increment experiments achieved 98.87% classification accuracy in static states, while utilizing less than 4.04% of the Intel AtomTM CPU. Furthermore, the 50ms embedded application provided a 46X speedup over an equivalent MATLAB implementation. The experiments showed fast response time for predicting the mode transitions. Lastly, this mobile CPU based design utilizes less power than other systems designed for similar applications, while still providing nearly 96% reserve to provide additional

expansion capability of our NMI. The results demonstrated the feasibility of a mobile CPU based real-time NMI for control of artificial legs.

Our future work includes utilizing the reserve capacity provided by this efficient implementation to provide real-time impedance based leg control [33, 34], real-time EMG motion artifact detection [35, 36], real-time EMG signal trust assessments [35, 36] and the development of a 20ms window increment NMI.

List of References

[1] Ziegler-Graham K, MacKenzie EJ, Ephraim PL, Travison TG, Brookmeyer R. Estimating the prevalence of limb loss in the United States: 2005 to 2050. Arch Phys Med Rehabil. 2008;10:422–429.

[2] Gailey, Robert, Kerry Allen, Julie Castles, Jennifer Kucharik, and Mariah Roeder, Review of secondary physical conditions associated with lower-limb amputation and long-term prosthesis use. Journal of Rehabilitation Research & Development. 2008.

[3] OSSUR, The POWER KNEE. [Online]. Available: http://www.ossur.com/prosthetic-solutions/bionic-technology/power-knee

[4] F. Sup, A. Bohara, and M. Goldfarb, “Design and control of a powered transfemoral prosthesis,” In. J. Rob. Res., vol. 27, no. 2, pp. 263-273, Feb. 1, 2008.

[5] Hargrove, Levi J., Ann M. Simon, Robert Lipschutz, Suzanne B. Finucane, and Todd A. Kuiken. Non-weight-bearing neural control of a powered transfemoral prosthesis. Journal of Neuroengineering and Rehabilitation. 2013.

[6] H.A. Varol, F. Sup, and M. Goldfarb, “Multiclass real-time intent recognition of a powered lower limb prosthesis,” IEEE Trans. Biomed. Eng., vol. 57, no. 3, pp. 542-51, Mar. 2010.

[7] H. Huang, F. Zhang, L. J. Hargrove, Z. Dou, D. R. Rogers, and K. B. Englehart, “Continuous locomotion-mode identification for prosthetic legs based on neuromuscular-mechanical fusion,” IEEE Trans Biomed Eng, vol 58, pp. 2867-75, 2011.

[8] H. Huang, T. A. Kuiken, and R. D. Lipshutz, “A strategy for identifying locomotion mode using surface electromyography,” IEEE Trans Biomed Eng, vol 56, pp. 67-73, 2009.

[9] F. Zhang, H. Huang, “Source Selection for Real-Time User Intent Recognition Toward Volitional Control of Artificial Legs,” IEEE Journal of Biomedical and Health Informatics, vol. 17, 2013

[10] Intel Corporation. (2010, June). “Intel® Atom™ Processor Z5xx Series Datasheet” [online]. Available: http://www.intel.com/content/www/us/en/processors/atom/atom-z540-z530-z520-z510-z500-45-nm-technology-datasheet.html [March 19, 2012].

[11] R. Hernandez, F. Zhang, X. Zhang, H. Huang and Q. Yang, “Promise of a Low Power Mobile CPU based Embedded System in Artificial Leg Control,” Conf Proc IEEE Engineering in Medicine and Biology (EMBC) 2012.

[12] Measurement Computing Corporation. (2008). “USB-1616HS-BNC User’s Guide” [online]. Available: http://www.microdaq.com/measurement_computing/documents/usb-1616hs-bnc-user-manual.pdf [May 21, 2012].

[13] AxiomTek Corporation. (2012). “Fanless Embedded System with Intel® Atom™ Processor” [online]. Available: http://axiomtek.com/Download/Spec/ebox530-820-fl.pdf [March 19, 2012].

[14] Intel Corporation. (2011, April). “Intel® Atom™ Processor Z6xx Series Datasheet” [online]. Available: http://www.intel.com/content/www/us/en/processors/atom/atom-z6xx-datasheet.html [March 19, 2012].

[15] D. Marr, F. Binns, D. Hill, G. Hinton, D. Kaoufaty, J. Moller, M. Upton “Hyper-Threading technology architecture and microarchitecture,” Intel Technology Journal, vol. 6 no. 1, pp 4-15, 2002.

[16] M. Weiskirchne. (2003, September). Comparison of execution times of Ada, C and Java. EADS Deutschland GmbH Military Aircraft. Muenchen, Germany. [online] Available: http://www.aicas.com/info/EADS_benchmark_languare_comparison.pdf [March 19, 2012].

[17] P. Sestoft. (2010, Feburary). Numeric performance in C, C# and Java. University of Copenhagen. Copenhagen, Denmark. [online] Available: http://www.itu.dk/~sestoft/papers/numericperformance.pdf [March 19, 2012].

[18] Cherrystone Software Labs. (2010, August). Algorithmic performance comparison between C, C++, Java and C# programming languages. Cherrystone Software Labs. Boston, MA [online] Available: http://www.cherrystonesoftware.com/doc/AlgorithmicPerformance.pdf [March 19, 2012].

[19] P. Ambika, H. Ahmed. (2001). A performance analysis of Java and C, University of Columbia, New York, NY, Avaliable: http://www.cs.columbia.edu/~sedwards/classes/2001/w4995-02/litsurveys/haseeb-ambika.pdf [March 19, 2012].

[20] D. Tiwari, S. Lee, J. Tuck, Y. Solihin. “Exploiting fine-grained parallelism in dynamic memory management,” IPDPS, 2010

[21] L. Ferres. (2010). Memory management in C: The heap and the stack. Universidad de Concepcion, Concepcion, Chile. [online] Avaliable: http://www.inf.udec.cl/~leo/teoX.pdf [March 19, 2012].

[22] A. Nicolau, “Loop quantization: unwinding for fine-grain parallelism exploitation,” Cornell Univerisy, 1985, Avaliable: http://ecommons.library.cornell.edu/bitstream/1813/6549/1/85-709.pdf [March 19, 2012].

[23] W. W. Hwu, P. P. Chang, “Inline function expansion for compiling C programs,” ACM SIGPLAN ’89 Conference on Programming Language Design and Implentation, Portland, Oregon, June 1989.

[24] J. L. Henessy and D. A. Patterson, “Pipelining: Basics and Intermediate Concepts,” in Computer Architecture: A Quantitative Approach 4th Edition, San Francisco CA, Morgan Kaufmann Publishers, 2007, pp. A21-A26.

[25] C. C. Chang, C. J. Lin, “LIBSVM: a library for support vector machnies,” ACM Transactions on Intelligent Systems and Technology, vol. 2 issue 3, pp. 27:1-27:27, 2011.

[26] F. Zhang, H. Huang, “Real-Time Recognition Of User Intent For Neural Control Of Artificial Legs,”MEC’11, New Brunswick, Fredericton, NB Canada, August 2011.

[27] X. Zhang, H. Huang, Q. Yang, “Design and implentation of a special purpose embedded system for neural machine interface," Conf Proc IEEE International Conference on Computer Design, 2010, pp. 166-72.

[28] R. Hernandez, and J. Faella, “Towards Policy and Guildelines for the Selection of Computational Engines,” Conf Proc IEEE Systems Conference 2013, Orlando, FL, April 2013.

[29] B. Furht, D. Grostick, et. al., ”Real-time UNIX systems design and application guide”, Kluwer Academic Publishers Group, Norwell, MA, USA

[30] Microsoft Corporation. Deveolpment Tools | Windows Embedded CE Tools for Developers. [online] Available: http://www.microsoft.com/ windowsembedded/en-us/develop/windows-embedded-ce-6-for-developrs.aspx [March 26, 2013]

[31] M. Barabanov, “A Linux-Based Real-Time Operating System”, M.Comp.Sci. Thesis, New Mexico Institue of Mining and Technology, Socorro, New Mexico, June 1, 1997

[32] Microsoft Corporation. (2012, February 7). Scheduling Priorities. [online] Available: http://msdn.microsoft.com/en-us/library/windows/desktop/ms685100(v=vs.85).aspx [March 19, 2012]

[33] F. Zhang, M. Liu and H. Huang, “Preliminary Study of the Effect of User Intent Recognition Errors on Volitional Control of Powered Lower Limb Prostheses,” Conf Proc IEEE Engineering in Medicine and Biology (EMBC) 2012.

[34] M. Liu, P. Datseris, and H. Huang, "A prototype for smart prosthetic legs: analysis and mechanical design," Conf Proc Proceeding of International Conference on Control, Robotics and Cybernetics. New Delhi, India: IEEE, March 21-23, Vol. 1. pp. 139-143, 2011

[35] Y. Liu, F. Zhang, Y. Sun, H. Huang, "Trust Sensor Interface for Improving Reliability of EMG-based User Intent Recognition", Conf Proc. IEEE Engineering in Medicine and Biology Society (EMBC) 2011, Boston, MA, Aug-Sept, 2011.

[36] X. Zhang, Y. Liu, F. Zhang, J. Ren, Y. Sun, Q. Yang, and H. Huang, "On Design and Implementation of Neural-Machine Interface for Artificial Legs," Industrial Informatics, IEEE Transactions on , vol.8, no.2, pp.418,429, May 2012

MANUSCRIPT 4

Design and Implementation of a Low Power Mobile CPU Based Embedded System for Artificial Leg Control

by

[1]Robert Hernandez, Qing Yang, He Huang, Fan Zhang,

and Xiaorong Zhang

is published in *the proceedings of the 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC'13)*,

Osaka, Japan, 2013. p. 5769-5772.

[1] Robert Hernandez, Qing Yang, He Huang, Fan Zhang, and Xiaorong Zhang, are with Department of Electrical, Computer, and Biomedical Engineering, University of Rhode Island, Kingston, RI, 02881, Email { rhernandez, qyang, huang, fzhang, zxiaorong }@ele.uri.edu.

Abstract

This paper presents the design and implementation of a new neural-machine-interface (NMI) for control of artificial legs. The requirements of high accuracy, real-time processing, low power consumption, and mobility of the NMI place great challenges on the computation engine of the system. By utilizing the architectural features of a mobile embedded CPU, we are able to implement our decision-making algorithm, based on neuromuscular phase-dependent support vector machines (SVM), with exceptional accuracy and processing speed. To demonstrate the superiority of our NMI, real-time experiments were performed on an able bodied subject with a 20ms window increment. The 20ms testing yielded accuracies of 99.94% while executing our algorithm efficiently with less than 11% processor loads.

4.1 Introduction

A pattern recognition (PR) strategy based on phase dependent and neuromuscular-mechanical fusion support vector machines (SVM) has been successfully developed in our research group to identify user intent in real-time to allow neural control of artificial legs [1, 2]. To make this strategy a feasible reality, a real-time neural machine interface (NMI) that is small, low cost, low power and capable of executing this computationally intensive algorithm needs to be developed. In our previous study we utilized FPGA technology to meet all of the NMI constraints with excellent results when executing a linear discriminant analysis (LDA) based classifier [3]. A non-linear SVM based algorithm was shown to provide increased accuracy over LDA [1], but is much more computationally intensive, which increases the complexity of an FPGA based design. This complexity exposes challenges such as language syntax, design environments, and toolsets during the design, implementation and troubleshooting phases of FPGA based systems [4].

Commodity mobile processors, such as the Intel Atom^{TM} Z530, are low power (2.2 watts [5]), low cost, and portable. Our prior offline study developed a prototype mobile processor based NMI to execute our complex PR algorithm and performed an offline study [6]. The study showed that a mobile processor based NMI had great promise in control of artificial legs [6]. However, in order to meet the special requirement of high accuracy and real time processing, tailoring our SVM based NMI software to this mobile PC architecture is desirable and challenging. We have developed fully functional software based on the SVM classifier on the mobile PC with all necessary interfaces for a data acquisition system with the capability to

acquire real-time electromyographic (EMG), mechanical force and moment data from human subjects. This newly developed NMI was combined with a Measurement Computing's USB-1616HS-BNC DAQ [7] to facilitate the collection of the real-time EMG and 6 degrees-of-freedom (DOF) mechanical data. This final NMI design was utilized to execute and test the performance of our phase dependent SVM based PR algorithm at a 20ms window increment during real-time experiments on an able bodied human subject.

This paper makes the following contributions:

- Design and implementation of a real-time capable NMI for artificial leg control based on a mobile processor;
- The first NMI embedded system to execute our phase dependent SVM based PR algorithm at 20ms window increments;
- A real time experiment that evaluates the potential use of mobile processors for real-time embedded implementation for neural control of powered lower limb prosthesis.

4.2 Software Design and Implementation

This study is based on a previously developed PR algorithm that identifies the user's locomotion mode based on electromyographic (EMG) signals acquired in real-time from thigh muscles and mechanical forces/moments signals acquired from 6 DOF load cell mounted on the prosthetic pylon [1,2]. The EMG and mechanical data are segmented by sliding analysis windows. Features data are extracted from raw EMG and mechanical signals in each analysis window and fused into a single feature vector. The feature vector is sent to a phase-dependent pattern classifier for determination of

user intent. The phase-dependent pattern classifier consists of four sub-classifiers, one for each individually defined gait phase. A gait phase detector identifies the current gait phase in real-time and selects the corresponding sub-classifier for final determination of user intent. A detailed description of this previously designed PR algorithm can be found in [1] and [2].

4.2.1 Feature Extraction

In this study, four time-domain (TD) features (the mean absolute value, the number of zero crossings, the waveform length, and the number of slope sign changes) were extracted from EMG signals in each analysis window. For mechanical data, the mean, minimum, and maximum values in each analysis window were extracted as the features. Further details on the feature extraction can be found in [1].

4.2.2 Phase Dependent Pattern Recognition

To accurately determine user intent, an SVM based classification architecture utilizing a Radial Basis Function (RBF) kernel and an SVM gamma parameter of 0.015 was used [1, 2]. The phase-dependent classifier is composed of four sub-classifiers corresponding to one of the following four gait phases: initial double limb stance (phase 1), single limb stance (phase 2), terminal double limb stance (phase 3), and swing (phase 4) [8]. Throughout this paper, inclusive of the figures, we utilize the following gait phase definitions: 1 - Initial Double Limb Stance, 2 - Single Limb Stance, 3 - Terminal Double Limb Stance and 4 - Swing. The gait phase detector uses the real-time vertical Ground Reaction Force (GRF) to determine the gait phases. In order to build the SVM sub-classifier models, a training procedure is conducted on all the acquired training data sets. During training phase, the output of the phase detector

is used to label the training data with its corresponding gait phase. Each sub-classifier is trained only with the data pertinent for its gait phase. During the real-time testing phase, the gait phase detector determines which sub-classifier is responsible for the determination of user intent. The gait phase detector's determination is used to select the appropriate sub-classifier to act upon the feature vector composed of fused EMG and mechanical data. The algorithmic data flow of the phase-dependent pattern recognition is shown in Fig. 1.2.

4.2.3 Software Architecture

We implemented the NMI software as shown in Fig. 1.2 in the C programming language. To meet real-time constraints, while executing on an Atom™ CPU, we implemented various performance enhancements techniques to the program. We took advantage of reduced dynamic memory management [9], loop unwinding [10] and inline function expansion [11].

To minimize the impacts of the real-time data logging on the application, a statically allocated and statically defined Random Access Memory (RAM) buffer was implemented that stored all the raw EMG, mechanical, classification and application performance data. The RAM buffer eliminated the need to write to the hard drive during time critical operations. Furthermore, it took advantage of the RAM's superior speed for storage. The real-time data logging for each classification was performed after all time-critical functions were completed (i.e., at the end of each classification). Lastly, the RAM buffer's contents were written to the hard drive for post analysis after the experiment was completed, such that no further time critical functions were being executed.

The final result is an embedded application specifically designed to minimize pipeline stalls, minimize OS impacts, minimize cost of memory allocation, minimize the impacts of real-time data logging and take advantage of the Intel Atom™ Z530 Processor hardware architecture. These enhancements provided the basis for the speed performance introduced by this embedded application.

As in our previous study [5], LIBSVM [12] was chosen as the open source library to utilize as the open source SVM libraries for our embedded application. This decision was based on LIBSVM's proven accuracy. Also, the analysis of LIBSVM's source code showed that it would be possible to modify the libraries for real-time use.

4.2.4 Software Implementation

To implement the Phase-Dependent PR algorithm, four applications were developed: a real-time training data capture application, a feature extraction & normalization application, a SVM training application and a Neuromuscular-Mechanical Fusion PR application. The real-time training data application captures training data for all the various human locomotion tasks. The feature extraction & normalization application accepts as input the real-time training data, performs the EMG and mechanical feature extraction and normalization, and then finally fuses the features into vectors. The feature vectors are then separated into their corresponding gait phases and provided to the training application. This application is also responsible for generating the normalization parameters required by the real-time PR application to normalize the real-time testing data, when determining user intent. The SVM training application accepts the four sets of training vectors and generates four SVM models, one model for each gait phase. The real-time PR application is used

during the real-time testing phase. It accepts as input: raw real-time testing data, the four gait phase SVM models, and the normalization parameters. The real-time PR application extracts EMG and mechanical features from the raw testing data acquired in real-time. The features are then fused and normalized, with the provided normalization parameters and formed into a vector. Finally, the application determines the current gait phase, and forwards the test vector to the respective phase based classifier for determination of user intent. The software implementation data flow is shown in Fig. 3.2.

4.3 Experimental Protocol

The AxiomTek eBOX530-820-FL1.6G fanless embedded hardware [13] with an Intel Atom^{TM} Z530 Processor [5] was chosen for the prototype design to test real-time feasibility and capability. To sample the raw EMG and mechanical data in real-time a Measurement Computing's USB-1616HS-BNC DAQ [7] system was interfaced with the AxiomTek embedded hardware. The Measurement Computing DAQ was chosen for its accuracy and capability to sample the data with a skew of 1 microsecond in between channels providing similar performance to that of a simultaneous sampling DAQ system.

A real-time performance evaluation utilizing a 20ms window increment with a window length of 160ms was conducted as part of this study. This experiment was conducted with approval of Institutional Review Board (IRB) at the University of Rhode Island and informed consent of the subject. The evaluations were performed on the data collected from a male able bodied subject. The collected data included the EMG signals from the subject's thigh muscles and mechanical forces/moments

measured by a 6 degree-of-freedom load cell mounted on the prosthetic pylon. The monitored muscles included the sartorius (SAR), rectus femoris (RF), vastus medialis (VM), adductor magnus (ADM), biceps femoris short head (BFS), biceps femoris long head (BFL), and semitendinosus (SEM).

The EMG and mechanical forces/moments were sampled at 1 KHz by the Measurement Computing's USB-1616HS-BNC DAQ device. The user intent decisions provided by the embedded hardware were provided via an analog output interface on the DAQ device. The experiment provided real-time gait-phase and user intent decisions to the console screen as a visual cue during the training and testing processes.

For all the experiment performed in this study, the prediction time will be defined as the total time to execute feature extraction, normalization, gait phase detection, majority vote and classification for a single analysis window.

4.4 Real-Time Performance Evaluation

The 20ms window increment embedded software design incorporated a real-time ten point majority vote algorithm as in [8] and the phase detector was tuned to the subject's locomotion patterns during the real-time training phase.

For this experiment, three tasks (level-ground walking (W), stair ascent (SA), and standing (ST)) and two mode transitions (ST→W and ST→SA) were studied. To ensure the subject's safety, the subject was allowed to use hand rails when necessary. To train the gait-phase classifier, the subject was instructed to perform each task for approximately 10 seconds. Two trials of standing data, three trials of walking data, and three trials of stair ascent data were accumulated to train the classifier. For the

real-time performance evaluation, 10 trials of each task and mode transitions were conducted (20 trials total). To assess the real-time performance of the NMI, the timing and processor loading of the application's execution on the embedded hardware are provided and the recognition accuracy of the NMI will be evaluated via a comparison with a similar LDA based NMI and the following parameters:

Classification Accuracy in the Static State: The static state is defined as the state where the subject has completed a transition and is continuously performing the same task (W, SA). The classification accuracy in the static state is the total number of correct classifications observed over the total number of classifications observed during the static state.

The Number of Missed Mode Transitions: The mode transition period starts from the beginning of gait phase 2 (single limb stance) and terminates at the beginning of gait phase 4 (swing). A mode transition is declared to have been missed, if no correct transition decision is made during this defined period.

Mode Transition Prediction Time: The mode transition prediction time is defined and the amount of time prior to the critical timing, during which the classifier user intent decision has stabilized and is no longer changing, such that safe switching of the prosthesis device is made possible. For this experiment, the critical timing is defined as the termination of the mode transition (i.e. - just prior to the start of the swing gait phase).

4.4.1 Recognition Accuracy of NMI and LDA Comparison

The overall classification accuracy of the NMI in the static states for all the predictions performed during the 20 trials inclusive of all tasks (W, SA, and ST) was

99.94%. No missed mode transitions were observed during the defined mode transition period. The mean mode transition prediction time for ST→SA was 658.0ms with a standard deviation of 155.6ms. The mean mode transition prediction time for ST→W was 534.0ms with a standard deviation of 103.3ms. The mode transition performance implies that user intent classification during transitions can be accurately determined, on the average, 514ms prior to the critical timing and be used for safe switching and control of the prosthesis. Representative trials, acquired during real-time testing, depicting the user intent classifications prior and during the ST→SA and ST→W transitions are provided in Fig. 4.1 and Fig. 4.2, respectively. As can be seen, the system is highly accurate and responsive. Furthermore, it can be seen that the

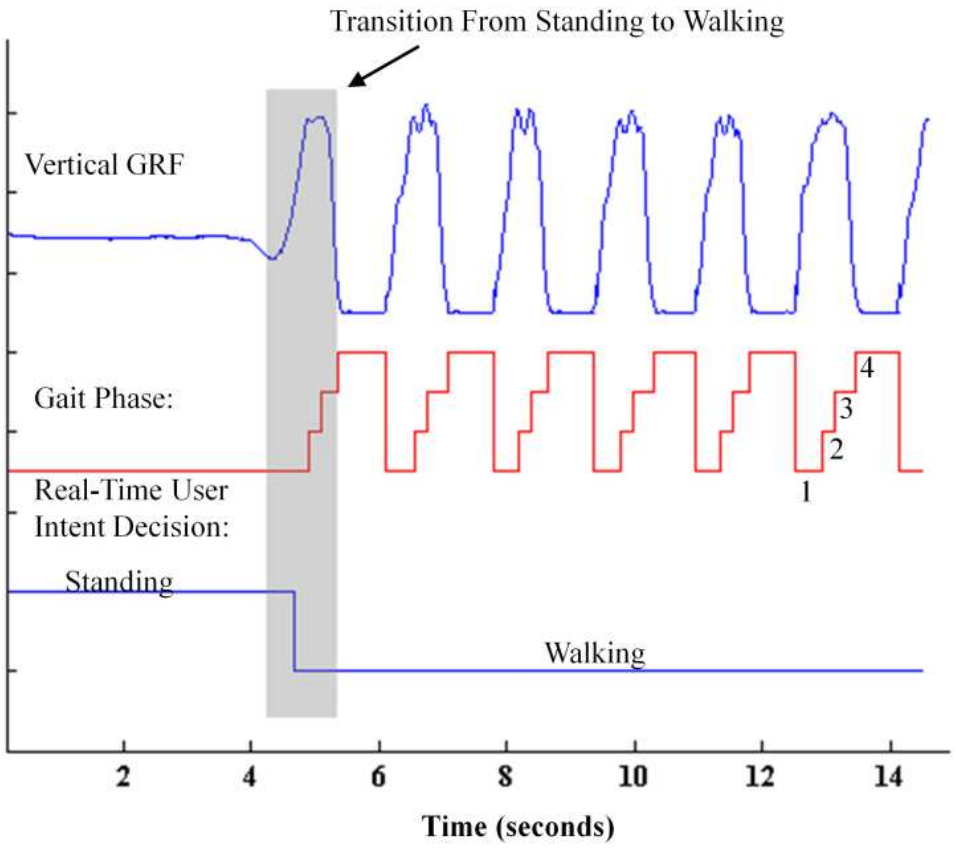

Figure 4.1. Real-Time Performance of a Standing to Walking Trial

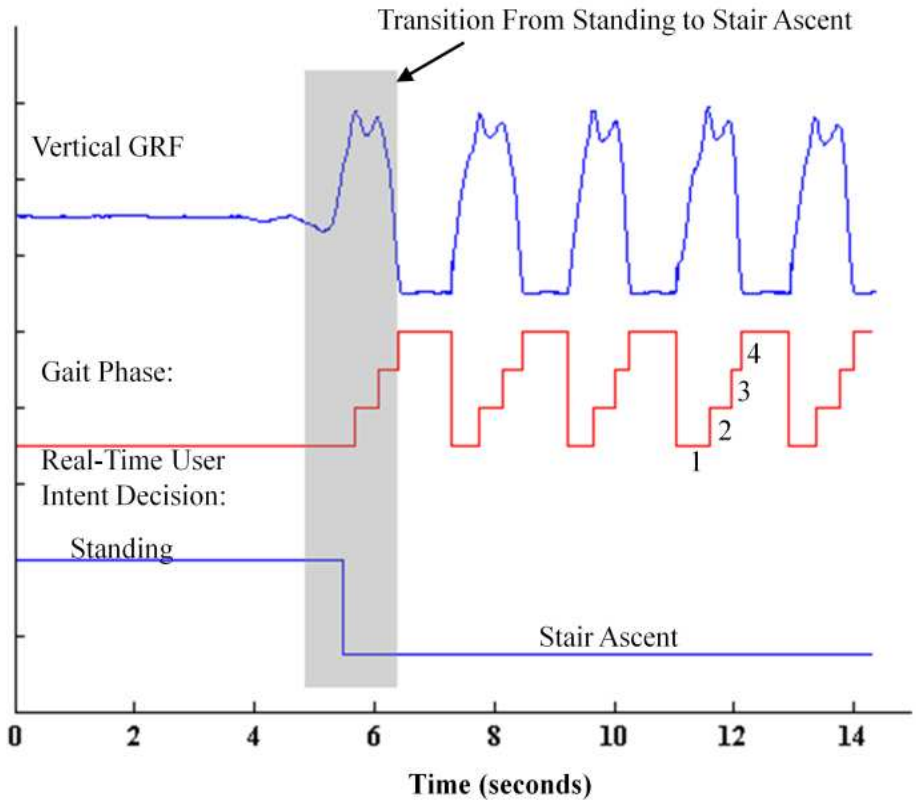

Figure 4.2. Real-Time performance of a Standing to Stair Ascent Trial

transitions were correctly predicted prior to the critical timing and the static state accuracy was 100% during these two trials.

In comparison, a LDA based neuromuscular-mechanical fusion, phase-dependent pattern recognition NMI provided 97.41% accuracy in the static states [3]. Similarly, the LDA study was based on the same three tasks (W, SA, and ST), utilized the same window increment of 20ms, the same window length of 160ms, and performed same number of trials as well.

4.4.2 Execution Timing and Processor Loading on the Embedded Hardware

A total of 14276 predictions were produced by the Intel Atom™ based embedded hardware during the trials. The mean prediction time per trial was 0.721ms

with a standard deviation of 0.0754ms. The worst case prediction executed in 2.124ms.

Due to the fact that there is additional loading on the CPU to execute the data logging for post analysis, the CPU loading provided by the operating system may be inaccurate; therefore the mean and maximum values of CPU loading were calculated using (4.1), which were 3.61% and 10.62% respectively.

$$\textbf{CPU Loading} = \frac{\textbf{Prediction Time}}{\textbf{Window Increment (20ms)}} * \textbf{100} \tag{4.1}$$

4.5 Conclusions

This paper presented the design and implementation of a mobile CPU based neural machine interface for artificial legs. The designed NMI prototype was tested on an able-bodied subject for classifying multiple movement tasks (level-ground walking, stair ascent and standing) in real-time. In the 20ms real-time window increment experiments, the system achieved 99.94% classification accuracy in static states, while utilizing less than 10.62% of the Intel Atom™ CPU. The experiment showed fast response time for predicting the mode transitions. Lastly, this mobile CPU based design utilizes less power than other systems designed for similar applications [6], while still providing nearly 90% reserve to provide additional expansion capability of our NMI. The results demonstrated the feasibility of a mobile CPU based real-time NMI for control of artificial legs.

Our future work includes utilizing the reserve capacity provided by this efficient implementation to provide real-time impedance based leg control, real-time EMG motion artifact detection, and a real-time EMG signal trust assessments; thereby

creating a single processor based NMI embedded solution that performs all these functions.

List of References

[1] H. Huang, F. Zhang, L. J. Hargrove, Z. Dou, D. R. Rogers, and K. B. Englehart, "Continuous locomotion-mode identification for prosthetic legs based on neuromuscular-mechanical fusion," IEEE Trans Biomed Eng, vol 58, pp. 2867-75, 2011.

[2] H. Huang, T. A. Kuiken, and R. D. Lipshutz, "A strategy for identifying locomotion mode using surface electromyography," IEEE Trans Biomed Eng, vol 56, pp. 67-73, 2009.

[3] X. Zhang, Q. Yang and H. Huang, "A Neural-Controlled Cyber Physical System for Intent Recognition for Artificial Legs," presented at Design Automation Conference, San Francisco, 2012 (Accepted).

[4] I. Gonzalez, E. El-Araby, P. Saha, T. El-Ghazawi, H. Simmler, S. Merchant, B. Holland, C. Reardon, A. George, H. Lam, G. Stitt, N. Alam, M. Smith, "Classification of application development for FPGA-based systems," Conf Proc National Aerospace Electronics Conference, 2008.

[5] Intel Corporation. (2010, June). "Intel® Atom™ Processor Z5xx Series Datasheet" [online]. Available: http://www.intel.com/content/www/us/en/processors/atom/atom-z540-z530-z520-z510-z500-45-nm-technology-datasheet.html [March 19, 2012]

[6] R. Hernandez, F. Zhang, X. Zhang, H. Huang and Q. Yang, "Promise of a Low Power Mobile CPU based Embedded System in Artificial Leg Control," Conf Proc IEEE Engineering in Medicine and Biology (EMBC) 2012.

[7] Measurement Computing Corporation. (2008). "USB-1616HS-BNC User's Guide" [online]. Available: http://www.microdaq.com/measurement_computing/documents/usb-1616hs-bnc-user-manual.pdf [May 21, 2012]

[8] F. Zhang, W. DiSanto, J. Ren, Z. Dou, Q. Yang, H. Huang, "A Novel CPS System for Evaluating a Neural-Machine Interface for Artificial Legs," Proceeding of 2nd ACM/IEEE International Conference on Cyber-Physical Systems, pp. 67-76, 2011.

[9] D. Tiwari, S. Lee, J. Tuck, Y. Solihin. "Exploiting fine-grained parallelism in dynamic memory management," IPDPS, 2010.

[10] A. Nicolau, “Loop quantization: unwinding for fine-grain parallelism exploitation,” Cornell University, 1985, Avaliable: http://ecommons.library.cornell.edu/bitstream/1813/6549/1/85-709.pdf [March 19, 2012]

[11] W. W. Hwu, P. P. Chang, “Inline function expansion for compiling C programs,” ACM SIGPLAN ’89 Conference on Programming Language Design and Implementation, Portland, Oregon, June 1989.

[12] C. C. Chang, C. J. Lin, “LIBSVM: a library for support vector machines,” ACM Transactions on Intelligent Systems and Technology, vol. 2 issue 3, pp. 27:1-27:27, 2011.

[13] AxiomTek Corporation. (2012). “Fanless Embedded System with Intel® Atom™ Processor” [online]. Available: http://axiomtek.com/Download/Spec/ebox530-820-fl.pdf [March 19, 2012]

CHAPTER 5

Conclusions and Future Work

5.1 Conclusions

This dissertation presented the research into the evolution of two small and low power architectural solutions for the University of Rhode Island (URI's) Support Vector Machines (SVM)-based Neural Machine Interface (NMI) algorithm. Manuscripts 1 thru 4 presented the offline research, Analysis of Alternatives (AoA), and the first two real time capable design iterations of the CPU-based architecture. At the time of publication, the research presented in Manuscript 1, showed that the mobile CPU based embedded system was URI's lowest power and smallest architectural solution capable of executing either URI's Linear Discriminant Analysis (LDA) or SVM-based NMI algorithm. The mobile CPU-based solution was further evolved via modifications to the NMI algorithm, such as a different choice of gamma for the SVM and a 20ms window increment, which ultimately led to URI's highest overall static prediction accuracy (99.94%), which was presented in Manuscript 4.

When comparing URI's algorithmic and architectural solutions to other published state of the art systems, intended to provide volitional control of powered lower limb prosthesis for transfemoral amputees, the URI solutions provide various contributions above and beyond that of the current state of the art in the fields of biomedical and computer engineering. These advantages are as follows:

- To the best of the author's knowledge, URI's architectural solutions, presented in Manuscripts 4, provides the highest published overall static

accuracy of any NMI, intended for artificial leg control, and tested to simultaneously classify multiple (more than three) distinct locomotion modes [1-19];

- In contrast to the intrinsic mechanical feedback systems described in [1-5], which appear to have difficulty in the development of a single model that can accurately classify more than two dynamic locomotion modes (e.g. - walk, stair up, stair down, ramp up, and ramp down) [6], as shown in Manuscripts 1 and 4, URI's NMI architectural solutions are capable of properly classifying a minimum of seven distinct locomotion modes;
- Unlike echo control based systems [7-10], which require instrumentation of the sound leg to determine the user intended locomotion modes [1-3, 6], URI's NMI architectural solution provides volitional control without the need to instrument the sound limb; instead URI's algorithm provides its volitional control via a much more natural method by sampling the neural commands sent by the brain to residual muscles in the amputated limb;
- To the best of the author's knowledge, URI's Mobile-CPU based architecture has the lowest power dissipation of any published NMI solution shown to be capable of accurately handling at least four simultaneous locomotion classes [1-6, 11-15].
- To the best of the author's knowledge, Manuscripts 1 thru 4 provide the only currently published C-based implementations of an SVM-based NMI algorithm, designed to utilize both mechanical and neural information,

optimized to enable real-time execution on small and low power architectures such as Digital Signal Processors (DSPs) and mobile-CPUs.

Based on the contributions above, URI's NMI solutions have been shown to provide many advantages over other state of the art powered lower limb prosthetic control algorithms and embedded architectures. URI's small and low power, architectural solutions are leading the way towards highly accurate volitional artificial leg control of powered prosthetic devices, thereby making a bionic leg a feasible reality in the near future.

5.2 Future Work

Although the research presented in this dissertation in a huge step towards making URI’s NMI algorithm a feasible reality, more research and development still needs to be performed in order to create a complete and final NMI solution. In particular it would be beneficial to add EMG anomaly detection and trust assessments to detect when the EMG signals have become unstable so the system can take appropriate action. This is beneficial in detecting and compensating for changes in EMG frequency and amplitude due to muscle fatigue, during workouts. It will also aid in detection of EMG contact failures due to dirt and sweat or simply a fallen EMG.

Furthermore, it is preferable that the final design provides impedance-based control of the artificial limb, rather than utilize a separate Finite State Machine (FSM) to perform this function. Lastly, it is desirable to further improve the accuracy of the NMI algorithm. One possible solution that may achieve higher accuracy is to provide an additional vote layer composed of two additional parallel classifiers, such as an

additional linear and a polynomial classifier, then use the output of the three classifiers to determine if a change in locomotion mode is to be permitted.

List of References

[1] Varol, H.A.; Sup, Frank; Goldfarb, M., "Multiclass Real-Time Intent Recognition of a Powered Lower Limb Prosthesis," Biomedical Engineering, IEEE Transactions on , vol.57, no.3, pp.542,551, March 2010.

[2] Sup, Frank; Varol, H.A.; Mitchell, J.; Withrow, T.J.; Goldfarb, M., "Self-contained powered knee and ankle prosthesis: Initial evaluation on a transfemoral amputee," Rehabilitation Robotics, 2009. ICORR 2009. IEEE International Conference on , vol., no., pp.638,644, 23-26 June 2009.

[3] Varol, H.A.; Sup, Frank; Goldfarb, M., "Real-time gait mode intent recognition of a powered knee and ankle prosthesis for standing and walking," Biomedical Robotics and Biomechatronics, 2008. BioRob 2008. 2nd IEEE RAS & EMBS International Conference on , vol., no., pp.66,72, 19-22 Oct. 2008.

[4] Ha, K.H.; Varol, H.A.; Goldfarb, M., "Volitional Control of a Prosthetic Knee Using Surface Electromyography," Biomedical Engineering, IEEE Transactions on , vol.58, no.1, pp.144,151, Jan. 2011

[5] Lawson, B.; Varol, H.A.; Huff, A.; Erdemir, E.; Goldfarb, M., "Control of Stair Ascent and Descent With a Powered Transfemoral Prosthesis," Neural Systems and Rehabilitation Engineering, IEEE Transactions on, vol.21, no.3, pp.466,473, May 2013

[6] Young, A.J.; Simon, A.M.; Hargrove, L.J., "A Training Method for Locomotion Mode Prediction Using Powered Lower Limb Prostheses," Neural Systems and Rehabilitation Engineering, IEEE Transactions on, vol.PP, no.99, pp.1,1

[7] Flowers, W.C.; Mann R.W., “An Electrohydraulic Knee-Torque Controller for a Prosthesis Simulator,” ASME J. Biomech Eng., vol.99, no.4,pp.3-8, 1977

[8] Stein, J.L.; Flowers, W.C., “Stance Phase Control of Above Knee Prostheses: Knee Control Versus Sach Foot Design,” ASME J. Biomech Eng., vol.20, no.1,pp. 19-28, 1987

[9] Grimes, D.L., “An Active Multi Mode Above Knee Prosthesis Controller,” Ph.D. Thesis, Dept. Mech. Eng., Massachussets Inst. Tech., Cambridge, MA, 1979

[10] Bedard, S.; Roy, P., “Actuated Leg Prosthesis for Above-Knee Amputees,” U.S. Patent 7,314,490, Jun. 17, 2003

[11] Ling, Chen; Qingsong, Ai; Yan, He; Quan, Liu; Wei, Meng, "A real-time leg motion recognition system by using Mahalanobis distance and LS_SVM," Audio, Language and Image Processing (ICALIP), 2012 International Conference on , vol., no., pp.668,673, 16-18 July 2012

[12] Zheng, Enhao; Chen, Baojun; Wang, Qining; Wei, Kunlin; Wang, Long, "A wearable capacitive sensing system with phase-dependent classifier for locomotion mode recognition," Biomedical Robotics and Biomechatronics (BioRob), 2012 4th IEEE RAS & EMBS International Conference on , vol., no., pp.1747,1752, 24-27 June 2012

[13] Chen, Baojun; Zheng, Enhao; Fan, Xiaodan; Liang, Tong; Wang, Qining; Wei, Kunlin; Wang, Long, "Locomotion Mode Classification Using a Wearable Capacitive Sensing System," Neural Systems and Rehabilitation Engineering, IEEE Transactions on , vol.21, no.5, pp.744,755, Sept. 2013

[14] Zheng, Enhao; Chen, Baojun; Wei, Kunlin; Wang, Qining, "Lower Limb Wearable Capacitive Sensing and Its Applications to Recognizing Human Gaits." Sensors 13, no.10, pp. 13334-13355, 2013.

[15] She, Qingshan; Luo, Zhizeng; Meng, Ming; Xu, Ping, "Multiple kernel learning SVM-based EMG pattern classification for lower limb control," Control Automation Robotics & Vision (ICARCV), 2010 11th International Conference on , vol., no., pp.2109,2113, 7-10 Dec. 2010

[16] Ming Meng; Zhizeng Luo; Qingshan She; Yuliang Ma, "Automatic recognition of gait mode from EMG signals of lower limb," Industrial Mechatronics and Automation (ICIMA), 2010 2nd International Conference on , vol.1, no., pp.282,285, 30-31 May 2010

[17] Hargrove, Levi J.; Simon, Ann M.; Young, Aaron J.; Lipschutz Robert D.; Finucane, Suzanne B.; Smith, Douglas G.; Kuiken, Todd A., “Robotic Leg Control with EMG Decoding in an Amputee with Nerve Transfers,” New England Journal of Medicine; vol.369, pp. 1237-1242, September 26, 2013

[18] Ceseracciu, E.; Reggiani, M.; Sawacha, Z.; Sartori, M.; Spolaor, F.; Cobelli, C.; Pagello, E., "SVM classification of locomotion modes using surface electromyography for applications in rehabilitation robotics," RO-MAN, 2010 IEEE , vol., no., pp.165,170, 13-15 Sept. 2010

[19] Hargrove, L.J.; Huang, H.; Schultz, A. E.; Lock, B.A.; Lipschutz, R.; Kuiken, T.A., "Toward the development of a neural interface for lower limb prosthesis control," Engineering in Medicine and Biology Society, 2009. EMBC 2009. Annual International Conference of the IEEE , vol., no., pp.2111,2114, 3-6 Sept. 2009